

Implications on the Learning of Programming Through the Implementation of Subsets in Program Development Environments

Peter J. DePasquale III

Dissertation submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

John A.N. Lee, Chairman
James D. Arthur
Joseph D. Chase
John A. Lewis
Manuel A. Pérez-Quñones

July 17, 2003
Blacksburg, Virginia

Keywords: programming environments, novice programmers, CS1, language subsets
Copyright 2003, Peter J. DePasquale III

Implications on the Learning of Programming Through the Implementation of Subsets in Program Development Environments

Peter J. DePasquale III

Abstract

The undergraduate Computer Science program at Virginia Tech is the largest in the Commonwealth of Virginia, of which a key component is “CS 1044: Introduction to Programming”, and is typical of a first course in computer programming throughout the USA. While the student access to learning resources has improved considerably with the development of web-based assets, students are still expected to use the same sophisticated program development tools as are used in industry. The perceived complexity of the learning environment currently in use drives many women and minority students from the Computer Science program. A great deal of attention has been paid to the need to administer the student assignments and the grading system for this course, so as to minimize the teaching/grading load, but little attention has been paid to the methodologies of learning the material through practice. The work reported herein is intended to improve the pedagogy of this course by creating and integrating teaching/learning tools that better manage the student’s engagement in the use of program development activities. Following the implementation of a three-element software system involving an interpreter for the C-language, a program development environment, and a data-monitoring/collection device, the system was deployed in support of the freshman course in parallel to the commercial system commonly used. The experiment concentrated on examining the impact of the simplified development environment and the effort required for students to complete assigned programming projects.

Acknowledgments

I am indebted to the assistance of my committee: *Dr. John A.N. Lee*, *Dr. James ‘Sean’ Arthur*, *Dr. Manuel A. Pérez-Quiñones*, *Dr. Joseph Chase*, and *Dr. John Lewis*. Each of them have remained supportive of my efforts and the desire to see this through. Special thanks to John Lewis for pushing me down this path. The idea to pursue the Ph.D. first came from him while I was working on my Master’s degree under his direction. I would not have considered this change in my life without John suggesting so. (OK John, not only had this better be worth it, I’m ready for this lifestyle you kept preaching about!) Thanks for allowing me to vent when I needed it, and thanks for the motivation over the last five years. This would not have been possible without your support.

The *CS1 Sandbox* programming environment would be nothing without the efforts of the student developers. Sure, they got credit for their work, but their insights into issues they experienced with programming environments as students was invaluable. *Logan Hanks* and *Jon Pryor* really helped bring the first fully functional version of the software to life and went beyond the call to manage the project in the earliest of stages. I do hope we gave them A’s for their efforts!

I owe special thanks to the Virginia Tech Statistical Consulting Center and *Robert Schulman*. Dr. Schulman was an invaluable help to my statistical questions, no matter how trivial. *William McQuain* deserves my appreciation for putting up with this experiment on his watch. Bill was instrumental in getting the students to buy into the experiment on the first day of class, and backed me up the whole way through.

For technical assistance both *Douglas Slotta* and *Philip Isenhour* receive my everlasting thanks. When Douglas first heard of my idea – to build an interpreter – he immediately pointed me toward the Purdue Compiler Construction Tool Set (PCCTS) as a needed tool. Though it had since been transformed into ANTLR he was correct in his assessment. Douglas was also noted for always pestering me at just the wrong time about mostly anything and was always there with a quick insult or demoralizing comment at the right time. Philip, simply put, is a Java God. When questions of implementation, debugging, or other annoyances arose, Philip had a solution in less than 20 minutes, or knew where to point me for the correct answer. His idea of the data collection system (socket connection to the server, dumped to a file, and a PHP-backend script to populate the database) worked beyond expectations. When in doubt about Java, go see Phil.

Finally, there is my advisor J.A.N. Lee. Though it took some convincing for him to accept me as his final student at Virginia Tech, I knew that I made the correct decision when I was talking with a group of former Ph.D. students from our department at SIGCSE in Austin, Texas (2000). When I told them who my advisor was, several replied “*Oooh, I wish I had picked J.A.N.!*” The rest nodded in agreement.

To say I learned a great deal from J.A.N. would be an understatement – few can claim their advisor introduced them to Dijkstra. J.A.N. has become a true friend who has kept me on the path, believed in me when I didn’t, and also encouraged my extra-curricular hobbies and frequent winter trips to Villanova (to see a certain team play basketball). His input on building the *CS1 Sandbox* has been both timeless and priceless (Lord knows he’s got enough experience with them!) His efforts in editing this document have been without equal. I have, at the very least, become a better writer. For all intents and purposes, my humble thanks and appreciation. Friday lunches will never be the same. Special thanks to Dee Lee for the occasions that J.A.N. and I worked at their home, and for welcoming me into their home like another son.

My time at Virginia Tech has not been without personal strife. With this degree completed, I move on to begin a career in teaching and computer science education research, which I have been eagerly awaiting. I offer my thanks to the rest of those who offered friendship and guidance along my journey.

This page intentionally left blank.

Contents

Acknowledgments	iii
List of Tables	xi
List of Figures	xv
1 Introduction	1
1.1 Learning to Program	3
1.1.1 Learning the Language of Starting a Project and of an Editor	4
1.1.2 Learning the Language of the Programming Paradigm	6
1.1.3 Learning the Language of the Programming Environment	7
1.2 Separation Between the Environment and the Compiler	8
1.3 Justification of Dissertation Topic	10
1.3.1 Identification of Problem	10
1.3.2 Research Questions	10
1.3.3 Proposed Solution	10
2 A Review of Previous Work	13
2.1 Previous Efforts in Subsetting Programming Languages	13
2.1.1 COBOL	13
2.1.2 BASIC	14
2.1.3 FORTRAN / Java	15
2.1.4 Ada	15
2.1.5 Microsoft .NET Common Language Specification	16
2.2 Older Pedagogic Programming Environments (Pre-1995)	16
2.2.1 The Cornell Program Synthesizer	16
2.2.2 Pascal Trainer	17

2.2.3	Emile	18
2.2.4	Solo	19
2.2.5	Bittitalk and View Matcher	23
2.2.6	Karel the Robot	24
2.3	Current Pedagogic Programming Environments (1995-2003)	26
2.3.1	DrScheme	26
2.3.2	DrJava	28
2.3.3	jGrasp	30
2.3.4	BlueJ	32
2.3.5	Ready to Program	34
2.3.6	Alice	35
2.3.7	CodeLab	37
2.3.8	ProgramLive	39
2.3.9	MiniJava	40
2.4	Summary	41
3	The <i>CS1 Sandbox</i> and Supporting Software	43
3.1	Overview	43
3.2	The <i>CS1 Sandbox</i> Programming Environment (client)	44
3.2.1	<u>A</u> N <u>o</u> t <u>h</u> e <u>r</u> <u>T</u> o <u>o</u> l <u>f</u> o <u>r</u> <u>L</u> an <u>g</u> u <u>a</u> g <u>e</u> <u>R</u> ec <u>o</u> gn <u>i</u> t <u>i</u> o <u>n</u> (ANTLR)	44
3.2.2	Input and Output	50
3.2.3	Graphical User Interface	52
3.2.4	Help System	54
3.2.5	Deployment of the Environment	55
3.3	Deployment of <i>CS1 Sandbox</i> Subset Information (server)	57
3.3.1	Subset Creation Tool	57
3.3.2	Downloading and Applying Subset Information	60
3.3.3	Enforcement of Subsets	60
3.4	Data Collection Mechanism	62
3.4.1	Client-side	62
3.4.2	Server-side	63
3.5	Debugging Support (for the <i>CS1 Sandbox</i> Developers)	64
3.5.1	Debugging Window	64
3.5.2	AST Viewer	66

4	The Design of the Experiment	67
4.1	The Purpose of the Experiment	67
4.2	The Classroom Environment	67
4.2.1	Closed Labs	68
4.2.2	Programming Projects	71
4.3	Approval of Experiment Participation	71
4.4	Subject Demographics	73
4.5	Division of Subjects into Experiment Groups	74
4.5.1	Programming Environment Usage Requirements	76
4.6	Data Collection	76
4.6.1	Electronic Data (Closed-Laboratory and Out-of-Laboratory Programming Projects)	77
4.6.2	Academic Scores	78
4.6.3	Post-Project Data Collection / Surveys	79
4.6.4	Error Recognition Experiment and Focus Group	79
5	The Results of the Experiments	81
5.1	Closed-Laboratory Data Collected	81
5.1.1	Closed-Laboratory Raw Scores	82
5.1.2	Number of Compilation / Execution Attempts	82
5.1.3	Number of Errors Produced	85
5.2	Out-of-Laboratory (Take-Home) Project Data Collected	86
5.2.1	Project Raw Scores	87
5.2.2	Number of Compilation / Execution Attempts	89
5.2.3	Number of Errors Produced	90
5.2.4	Overall Time-on-Task	92
5.2.5	Post-Project Survey Results	93
5.3	Error Recognition Results	107
5.4	Focus Group Transcript	108
6	Analysis of the Collected Data	117
6.1	Statistical Methods Used in Analysis	117
6.1.1	Chi-Squared	117
6.1.2	Analysis of Variance (ANOVA) & Duncan's Procedure	118
6.1.3	Kruskal-Wallis	118

6.1.4	Repeated Measures	119
6.2	Statistical Tests for Parity of the Distribution of Experiment Subjects	119
6.2.1	Analysis of Experiment Group Parity - Categorical	119
6.2.2	Analysis of Experiment Group Parity - Quantitative	120
6.2.3	Demographic Conclusions	122
6.3	Statistical Evaluation of Closed-Laboratory Results	123
6.3.1	Analysis of Closed-Laboratory Scores	123
6.3.2	Analysis of Closed-Laboratory Compilation / Execution Attempts	126
6.3.3	Analysis of Closed-Laboratory Compilation / Execution Errors	128
6.3.4	Closed-Laboratory Summary	129
6.4	Statistical Evaluation of Out-of-Laboratory Project Results	131
6.4.1	Analysis of Out-of-Laboratory Project Scores	132
6.4.2	Analysis of Out-of-Laboratory Project Compilation / Execution Attempts	134
6.4.3	Analysis of Out-of-Laboratory Project Compilation / Execution Errors	136
6.4.4	Analysis of Out-of-Laboratory Project Time-On-Task	137
6.4.5	Analysis of Out-of-Laboratory Post-Project Survey Results	138
6.4.6	Observations from the Post-Project Survey Write-In Comments	151
6.4.7	Out-of-Laboratory Summary	152
6.5	Statistical Evaluation of Error Recognition	153
6.5.1	Analysis of Error Recognition Experiment Results	153
6.5.2	Error Recognition Summary	155
6.6	Observations from the Focus Group Transcript	155
7	Conclusions, Contributions, Observations, and Future Work	157
7.1	Conclusions	157
7.1.1	Impact of the Use of a Simplified Environment	157
7.1.2	Quantitative Impact of Subsets on User Effort	158
7.2	Contributions of this Work	159
7.3	Observations from this Work	159
7.3.1	Implications on Programming Environments	160
7.3.2	Implications on the Syllabus of CS1 Courses	161
7.4	Future Work	162
	Bibliography	165

A IRB Application and Student Agreement	171
B Demographic Survey	181
C CS1044 Course Syllabus – Fall 2002	183
D Closed-Laboratory Specifications	187
D.1 Laboratory #1	188
D.2 Laboratory #2	190
D.3 Laboratory #3	193
D.4 Laboratory #4	195
D.5 Laboratory #5	198
D.6 Laboratory #6	200
D.7 Laboratory #7	202
D.8 Laboratory #8	204
D.9 Laboratory #9	206
D.10 Laboratory #10	208
D.11 Laboratory #11	210
D.12 Laboratory #12	212
D.13 Laboratory #13	214
E Out-of-Laboratory Project Specifications	217
E.1 Project #1	218
E.2 Project #2	221
E.3 Project #3	225
E.4 Project #4	228
E.5 Project #5	236
F Post-Project Survey(s)	243
G Error Recognition Experiment Handout	245
H Subset Configuration File	253
I Grammar Recognized by the <i>CS1 Sandbox</i>	259

J ANTLR Input and Output Files	271
J.1 ANTLR Input File Used to Create <i>CS1 Sandbox</i> 's Lexer (CLexer.g)	272
J.2 ANTLR-produced Output File for the <i>CS1 Sandbox</i> 's Lexer (CLexer.java)	278
J.3 ANTLR Input File Used to Create <i>CS1 Sandbox</i> 's Parser (CParser.g)	307
J.4 ANTLR-produced Output File for the <i>CS1 Sandbox</i> 's Parser (CParser.java)	321
J.5 ANTLR Input File Used to Create <i>CS1 Sandbox</i> 's Semantic Validator (Semantic.g)	406
J.6 ANTLR-produced Output File for the <i>CS1 Sandbox</i> 's Semantic Validator (Semantic.java)	437
J.7 ANTLR Input File Used to Create <i>CS1 Sandbox</i> 's Interpreter (Interp.g)	498
J.8 ANTLR-produced Output File for the <i>CS1 Sandbox</i> 's Interpreter (Interp.java)	524
K Vita	567
Index	573

List of Tables

1.1	Program Development Environment Domains	3
2.1	Summary of capabilities of the programming environments presented in this dissertation. . .	42
3.1	Student Developer Population By Semester	43
3.2	Pre-Processing Directives Supported by the <i>CS1 Sandbox</i>	47
3.3	Help File Identifier Ranges and Descriptions	56
4.1	CS1044 Section Breakdown, Fall 2002	67
4.2	Grade Weightings for CS1044	68
4.3	Schedule of topic coverage for CS1044	69
4.4	Summary of CS1044 Closed-Laboratory Assignments	69
4.5	Summary of CS1044 Out-of-Laboratory (Take-Home) Programming Projects	72
4.6	Demographic Distribution of the Experiment's Subjects	74
4.7	Demographic Distribution and Means of Freshmen (Experiment Subjects and Survey Respondents)	75
4.8	Distribution of Subjects and Programming Environments	75
4.9	Subset Release Sequence	77
5.1	Mean Closed-Laboratory Score Details	82
5.2	Mean Closed-Laboratory Compilation Details	84
5.3	Mean Closed-Laboratory Compilation Errors	85
5.4	Number of subjects completing each out-of-laboratory project	87
5.5	Mean Out-of-Laboratory Project Score Details (projects #1 through #5).	88
5.6	Mean Out-Of-Laboratory Compilation Details	89
5.7	Mean Out-Of-Laboratory Compilation Error Details	91
5.8	Mean Out-Of-Laboratory Time-On-Task for Programming Projects	92

5.9	Self-Reported Project Time-On-Task	94
5.10	Self-Reported Project Number of Compilations/Executions	95
5.11	Mean rating values from the post-project survey question #3	96
5.12	Mean rating values from the post-project survey question #4	97
5.13	Mean rating values from the post-project survey question #5	98
5.14	Mean rating values from the post-project survey question #6	99
5.15	Mean rating values from the post-project survey question #10	100
5.16	Frequency of responses to the write-in post-project survey questions	102
5.17	Selected Write-In Comments from the Post-Project Surveys - .NET Users	103
5.18	Selected Write-In Comments from the Post-Project Surveys - Sandbox Without Subsets Users	104
5.19	Selected Write-In Comments from the Post-Project Surveys - Sandbox With Subsets Users .	105
5.20	Selected Write-In Comments from the Post-Project Surveys - Migrated Users	106
5.21	Error Experiment Classification Breakdown and Frequency.	107
5.22	Error Experiment Data Collected	107
6.1	Chi-Squared Test of Subject Groups vs. Class Standing	120
6.2	Chi-Squared Test of Subject Groups vs. Gender	120
6.3	Chi-Squared Test of Subject Groups vs. Exiting High School	121
6.4	Chi-Squared Test of Subject Groups vs. Age	121
6.5	Analysis of Variance (ANOVA) Tests of Numeric Demographic Results	122
6.6	Summary of significant findings from the statistical tests on demographic data.	122
6.7	ANOVA Test Results for Mean Closed-Laboratory Scores	124
6.8	Duncan's multiple range test – closed-laboratory scores from laboratories #1 and #6	124
6.9	ANOVA Test Results for Mean Closed-Laboratory Compilations	126
6.10	ANOVA Test Results for Mean Closed-Laboratory Compilation Errors	129
6.11	ANOVA Test Results Mean Out-of-Laboratory Project Scores	132
6.12	Repeated Measures Test Results for Mean Out-Of-Laboratory Compilations	135
6.13	Repeated Measures Test Results for Mean Out-Of-Laboratory Compilation Errors	137
6.14	Repeated Measures Test Results for Mean Out-Of-Laboratory Time-On-Task	138
6.15	Mean Out-Of-Laboratory Time-On-Task Percentage for Programming Projects	139
6.16	Self-Reported Project Time-On-Task	140
6.17	Self-Reported Project Number of Compilations/Executions	141
6.18	Kruskal-Wallis analysis results from the post-project survey question #3	143
6.19	Pair-wise Kruskal-Wallis analysis results from post-project surveys #4 & #5	144

6.20	Kruskal-Wallis analysis results from the post-project survey question #4	145
6.21	Pair-wise Kruskal-Wallis analysis results from post-project surveys #1, #2, and #3	146
6.22	Kruskal-Wallis analysis results from the post-project survey question #5	147
6.23	Pair-wise Kruskal-Wallis analysis results from post-project surveys #2 and #3	147
6.24	Kruskal-Wallis analysis results from the post-project survey question #6	149
6.25	Kruskal-Wallis analysis results from the post-project survey question #10	150
6.26	Pair-wise Kruskal-Wallis analysis results from post-project surveys #1, and #3	150
6.27	Chi-Squared Test of Subject Groups vs. Recognition of Syntax Errors	153
6.28	Chi-Squared Test of Subject Groups vs. Recognition of Semantic Errors	154
6.29	Chi-Squared Test of Subject Groups vs. Recognition of Logic Errors	154
6.30	Chi-Squared Test of Subject Groups vs. Recognition of Source Code Without Errors	154
6.31	Summary of key error recognition experiment performance (experiment groups combined) . .	155

This page intentionally left blank.

List of Figures

1.1	The project build inquiry dialog from the Microsoft Visual C++ development environment	5
1.2	The Microsoft Visual C++ .NET programming environment	5
1.3	A view of traditional GUI-based programming environments	8
2.1	The Pascal Trainer’s interface being used to select a module of exercises	18
2.2	The Pascal Trainer providing an exercise for a student	19
2.3	A view of a new, empty project in the Emile environment.	20
2.4	A project in Emile which simulates two-dimensional projectile motion	21
2.5	An example of a data structure in the Solo language	21
2.6	A sample help sequence in the Solo programming environment.	22
2.7	The JKarelRobot programming environment	25
2.8	The JKarelRobot programming environment’s instant interface	26
2.9	The DrScheme programming environment	27
2.10	The DrJava programming environment	29
2.11	The jGrasp environment displaying a control structure diagram	30
2.12	The jGrasp environment displaying a corresponding UML diagram	31
2.13	The jGrasp environment displaying a corresponding Complexity Profile Graph	32
2.14	The BlueJ Java program development/visualization environment	33
2.15	Interacting with an instantiated object in BlueJ	33
2.16	The Ready To Program development environment for Java	35
2.17	The Alice programming environment	36
2.18	A view of a sample 3-D “world” created by the Alice programming environment	36
2.19	Attempting a problem in the CodeLab programming environment	38
2.20	Obtaining an exercise result from the CodeLab programming environment	38
2.21	Obtaining an error message from the CodeLab programming environment	39

3.1	ANTLR role in processing a grammar file	45
3.2	An architectural view of the <i>CS1 Sandbox</i> programming environment (client-side)	46
3.3	An example of a semantic validation of the <code>if</code> programming statement as input to ANTLR	48
3.4	An example of interpretation directives of the <code>if</code> programming statement as input to ANTLR	49
3.5	The calling sequence of ANTLR within the <i>CS1 Sandbox</i>	50
3.6	The <i>CS1 Sandbox</i> Programming Environment's input/output interface	51
3.7	The <i>CS1 Sandbox</i> Programming Environment	52
3.8	The <i>CS1 Sandbox</i> Programming Environment's About window	53
3.9	The <i>CS1 Sandbox</i> Programming Environment showing several errors	55
3.10	A Sample <i>CS1 Sandbox</i> Help Window	57
3.11	The Subset Creation Tool's interface used to create language subsets	58
3.12	The Subset Creation Tool's interface used to distribute subset configuration files	59
3.13	The interface used to select from which server and port to attempt a subset download	60
3.14	The interface used to select which subset configuration file to download	60
3.15	The <i>CS1 Sandbox</i> interface used to display the current status of language constructs/elements	61
3.16	Data Submission Format	63
3.17	A sample data submission from a subject's programming environment	64
3.18	A pictorial representation of the entire <i>CS1 Sandbox</i> data collection system	65
3.19	The internal debugging interface	65
3.20	The Abstract Syntax Tree (AST) display interface	66
5.1	Graph of mean closed-laboratory scores (laboratories #1 – #13)	83
5.2	Graph of mean number of closed-laboratory compilations (laboratories #1 – #9)	84
5.3	Graph of mean number of closed-laboratory compilation errors (laboratories #1 – #9)	86
5.4	Graph of mean out-of-laboratory project scores (projects #1 – #5)	88
5.5	Graph of mean number of out-of-laboratory project compilations (projects #1 – #3)	90
5.6	Graph of mean number of out-of-laboratory project compilation errors (projects #1 – #3)	91
5.7	Graph of mean time-on-task (in seconds) of out-of-laboratory projects (projects #1 – #3)	92
5.8	Graph of mean self-reported number of days on task for the out-of-laboratory programming projects	95
5.9	Graph of mean self-reported number of compilation and execution attempts for the out-of-laboratory programming projects	96
5.10	Graph of mean survey ratings of difficulty in starting a new project/program	97
5.11	Graph of mean survey ratings of difficulty in entering/debugging source code	98

5.12	Graph of mean survey ratings of difficulty in testing project solution	99
5.13	Graph of mean survey ratings of project difficulty	100
5.14	Graph of mean survey ratings of the frequency of use of built-in help	101
6.1	Graph of mean closed-laboratory scores (laboratories #1 – #13) with data trends	125
6.2	Graph of mean number of closed-laboratory compilations (laboratories #1 – #9) with data trend	127
6.3	Graph of mean number of closed-laboratory compilation errors (laboratories #1 – #9) with data trend	130
6.4	Graph of mean out-of-laboratory project scores (projects #1 – #5) with data trends	133
6.5	Graph of mean out-of-laboratory project scores (projects #1 – #5) with data trends (with both zero project scores removed)	134
6.6	Graph of mean number of out-of-laboratory project compilations (projects #1 – #3) with data trend	135
6.7	Graph of mean number of out-of-laboratory project compilation errors (projects #1 – #3) with data trend	136
6.8	Graph of mean time-on-task (in seconds) of out-of-laboratory projects (projects #1 – #3) with data trend	139
6.9	Graph of mean self-reported days on task with data trend	140
6.10	Graph of mean self-reported number of compilation/execution attempts with data trend	142
6.11	Graph of mean survey ratings of difficulty in starting a new project/program with data trends	143
6.12	Graph of mean survey ratings of difficulty in entering / debugging source code with data trends	145
6.13	Graph of mean survey ratings of difficulty in testing project solution with data trends	148
6.14	Graph of mean survey ratings of project difficulty with data trends	149
6.15	Graph of mean survey ratings of the frequency of use of built-in help with data trends	151

This page intentionally left blank.

Chapter 1

Introduction

Within the computer science community, there has recently been an examination of programming languages and integrated development environments (IDEs) for use in conjunction with teaching CS1 courses (introductory programming courses which generally follow the syllabus for “CS1” within the IEEE/ACM-CS curriculum of 1978 [6]¹) to novice programmers. What is most striking about these teaching/learning environments is that for the most part they seek to follow one of the following approaches:

- the creation of a simplified language in which the student learns to program (Logo [45, 59], Euphoria [62], and Karel [61]), with the idea of ultimate migration to a more mainstream (commercially successful) language with advanced features;
- the simplification of the development environment in order to simplify the code development process [38]; or
- the development of an environment that supports code development and visualization of the authored code [18, 40].

Ironically what these approaches seemingly lack in their development environments is a similarity to the methods commonly used to teach introductory programming classes. When first introduced to a programming language, novice programmers generally are introduced to simple, stand-alone concepts that can be easily digested and understood. These concepts are then built upon in an iterative manner as additional language elements are introduced.

In essence, what educators are doing in the classroom is subsetting a programming language and introducing successive supersets to the student population. Many novice C programming students around the world are introduced to their first computer program with code similar to the following:

```
void main () {  
    printf("Hello world!");  
}
```

¹The IEEE/ACM Computing Curriculum of 2001 [64] continues to acknowledge the use of the “CS1” moniker for these introductory programming courses. However, in revising the curriculum, the 2001 guidelines utilize a course numbering scheme of CS1XYz – where ‘X’ is subject area designator, ‘Y’ is a identifying number within a subject area, and ‘z’ is a particular pedagogical approach (procedural, object-oriented, functional, etc.) to refer to the introductory computer science courses.

Here, the student is introduced to the most simple of computer programs. Only the bare bones concepts of a main function and system output are introduced. Generally, additional components of the language are introduced over the course of the term. By the end of the term, the students gain exposure to a majority of the code of the programming language and usually have the ability to author programs given only program specifications.

As they learn to implement programs, many students use either a simplified programming environment, a full-strength command line compilation system, or a programming environment with an visual interface and compiler designed for professional application development.

Michael de Raadt [23] recently found that 45% of respondents in a census of Australian Universities did not use a programming environment in their introductory programming courses.² The prevalent reasons cited by de Raadt included:

- the cost for the students,
- the time required to familiarize students with their environments, and
- the blurring of distinct steps in the programming process.

Despite the availability of professional environments, de Raadt concluded that:

“...there is a lack of tools that are designed specifically for novice programmers, are freely available, easy to use, do not obscure the details of the programming process, and in which instructors can be confident in teaching.”

Are CS1 students developing full-content applications? Do they need full strength debuggers, project management tools (such as source code control tools and profilers)? In many cases, these extraneous tools are never covered as part of the curriculum in novice programming classes and their presence in the programming environment only serves to confuse novice programmers (if examined at all). To make matters worse, it may be assumed in the courses that follow CS1 that these tools were covered as part of the curriculum.

I hypothesize that novice students do not need such complex tools and that the introduction of a highly complex structured programming environment to these students is detrimental to a clear understanding of their first programming language and programming in general. This sentiment was echoed by Russ Bjork in a March 2003 posting to the Association for Computing Machinery (ACM) Special Interest Group for Computer Science Education (SIGCSE) LISTSERV where he stated:

“I noticed a recent reply to this thread from Joe Bergin which included an “ad” for the Karel++ book he co-authored. I have used the Java version of this book in teaching an objects-first approach to Java for four years, and would strongly endorse using an approach like this to start students off. Using a micro-world in which language overhead is minimized allows students to focus on the higher level issues at the outset. Without something like this, there’s just too much “stuff” involved in producing visible results from initial programs, IMHO. We spend four classes, two closed labs, and a project on Karel - which I believe to be time well spent in terms of students being able to grasp the concept of objects and methods as well as selection and repetition when we meet these topics again later in the course.” [11]

²A less formal survey conducted by this author of members of the ACM SIGCSE LISTSERV in the spring of 2002 indicated that 20% of respondents indicated that their CS1 curriculum used GCC (a command line compiler) and nearly 75% used a programming environment with a graphical user interface.

Their first development environment should be simple to use, provide clear consistent error and warning messages (which do not confound the student), *and* should grow with them in a manner similar to the pedagogical methodology that the syntax of the language is being introduced in the classroom. This dissertation focuses on the argument that the students would be better suited to work in the domain that is tailored to education (subsetting and non-professional) (see Table 1.1 for a pictorial representation).

Table 1.1: Program Development Environment Domains

	Environments	
	Educational	Commercial
Subsetting (staged) language implementation	Best for students?	None known
Complete language implementation	Many exist	Many exist

1.1 Learning to Program

From the outset, the programming of a computer has been detailed as a non trivial task. In his memoirs [89], Maurice Wilkes recollects

“By June 1949 people had begun to realize that it was not so easy to get a program right as had at one time appeared. I well remember when this realization first came on me with full force. The EDSAC was on the top floor of the building and the tape-punching and editing equipment one floor below on a gallery that ran round the room in which the differential analyser was installed. I was trying to get working my first non-trivial program, which was one for the numerical integration of Airy’s differential equation. It was on one of my journeys between the EDSAC room and the punching equipment that “hesitating at the angles of stairs” the realization came over me with full force in that a good part of the remainder of my life was going to be spent in finding errors in my own programs.”

Despite the advances of time in computing (the move from tape entry to keyboard entry, the use of programming environments to aide in the development of large scale programs, etc.) the notion of learning to program has apparently not made easier.

In his landmark paper, *No Silver Bullet* [13], Fredrick Brooks states

“I believe the hard part of building software to be the specification, design and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation. We still make syntax errors, to be sure; but they are fuzz compared to the conceptual errors in most systems.

If this is true, building software will always be hard.”

To novice programmers, this last statement is particularly true. In their first term of programming class, a novice programmer will likely encounter difficulties with all of the points that Brooks cites (even those he categorizes as fuzz). Apart from understanding the algorithm design process and algorithm creation, novice programmers are required to master three fundamental abstract languages in their journey:

- the languages of establishing/initiating a project and of an editor,

- the language of the programming paradigm with which they will be working, and
- the language of the programming environment to compile and run their source code.

Each of these are discussed in more detail below.

1.1.1 Learning the Language of Starting a Project and of an Editor

Using a program development environment can be a highly challenging task. Learners must face the task of comprehending the paradigm that dictates how source code files are organized and managed within the environment. Understanding how to begin working with the environment in a simple fashion is a key element to promoting success in a novice student.

Throughout the course of the spring semester of 2001, this author ran a pilot study which monitored and obtained feedback from CS1044–Introduction to Programming in C (the Virginia Tech version of CS1) students using Microsoft Visual C++ (version 6.0)³ [53] at Virginia Tech. The investigation involved seven volunteer subjects, encompassing their use of the programming environment through the entire term. The following tasks were performed by this author during the course of the study:

1. The subjects' first three uses of Microsoft Visual C++ capturing the participants' emotions, comments, and thoughts of initially using a production-level programming environment. Roughly 85% (six) of the group had not used Microsoft Visual C++ prior to the study and the sole remaining subject had experience from taking CS1044 the previous term (as the subject was re-taking the course again).
2. Monthly group meetings comprising myself and the subjects where held to obtain feedback of problem areas they encounter with the Microsoft Visual C++ environment, the C language and their experience in CS1044 in general.
3. The subjects were asked to keep a diary of syntax problems and error messages that they encountered while completing their course projects. Diaries were reviewed every month following the initial videotaped sessions being completed to monitor their progress and to extract comments from the diaries.
4. I conducted surveys (pre/post) on their background with programming, and experiences programming in C with Microsoft Visual C++.

While the project management abstraction might seem trivial to experienced software developers (or even to novice programmers at the end of their first course in programming), nearly every one of the students in the above noted study remarked that they were challenged by the startup procedure to author their code. One student went so far as to suggest a wizard (a Microsoft help system aid) for project creation.

Microsoft Visual C++ (version 6.0), required the creation of a project workspace (an abstract project management construct) to successfully create an executable application. Its creation can be completed following the startup sequence of the development environment, but it can also be deferred until a source code file is to be compiled (and thus needs to be saved and added to a project). When presented with the latter situation, the user is shown the dialog window seen in Figure 1.1. The learner in this situation may not immediately recognize that such a message is enforcing the project management's paradigm and that it is not truly not an insurmountable problem, but it helps skip the need to startup a project by the user.

³Visual C++ and .NET are registered trademarks of Microsoft Corporation.

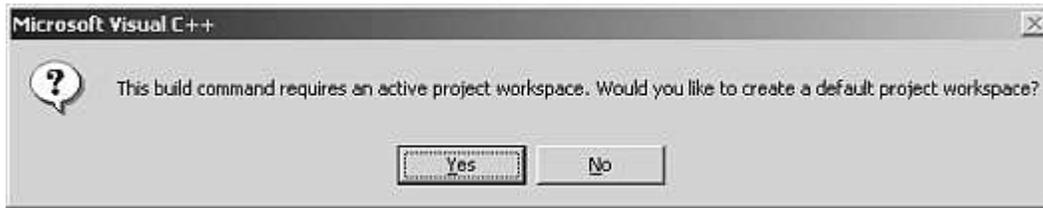


Figure 1.1: The project build inquiry dialog from the Microsoft Visual C++ development environment (version 6.0)

The Microsoft Visual C++ .NET [52] (hereafter referred to as .NET)⁴ interface modifications, though more aesthetically pleasing, are no less daunting to the novice. The initial interface appears in Figure 1.2.

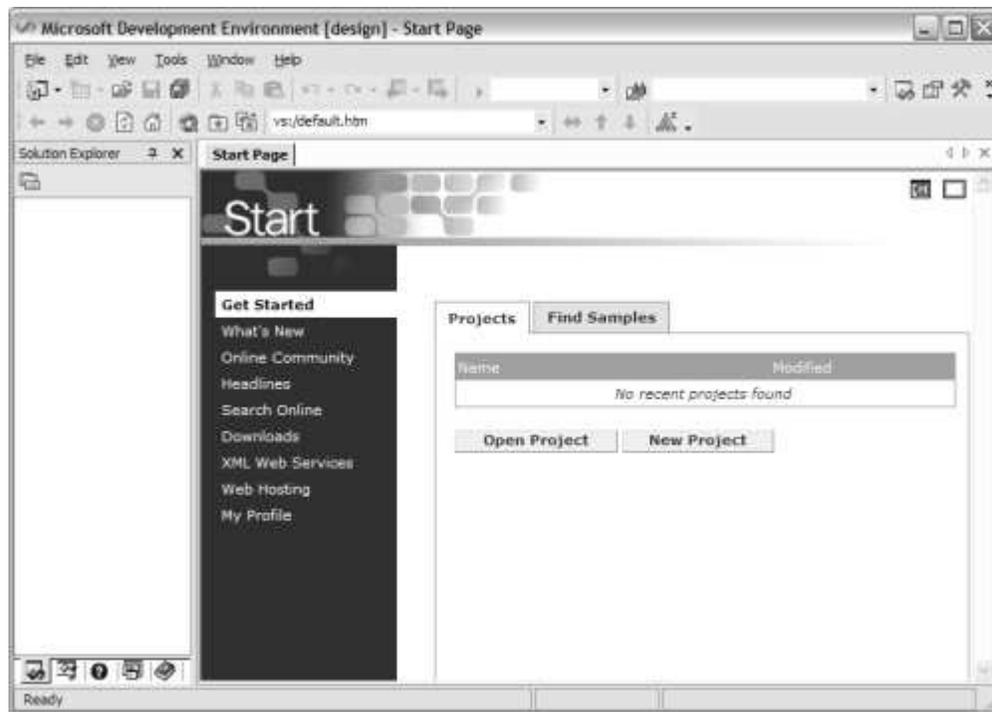


Figure 1.2: The Microsoft Visual C++ .NET programming environment. The interface contains numerous buttons and menu items. It also includes a panel for managing the project (labeled as a Solution) and a wizard (shown as the Start Page in the lower right region of the interface to assist in starting the management of a new application (type of application, organization, etc.).

Source code editors are another source of potential problems for novice students. Graphical User Interface (GUI)-based program development environments commonly rely on the WIMP (Window, Icon, Menu,

⁴The .NET programming environment became the de facto programming environment used in the department's curriculum (starting with the CS1044 and CS1104 (Introduction to Computer Science) courses) in the Fall of 2002.

Pointing device ⁵) model for interaction between the user and the environment. While it is a fairly common model, the WIMP paradigm can befuddle and confuse extreme novice computer users (and programmers), forcing the learner to master the language of WIMP first to accomplish their programming task.

This hurdle is less likely to arise in the future with the increasing proliferation of personal computers in the home and school. An August 2000 Department of Commerce study [56] stated that nine of ten school-age children (six to seventeen years old) had access to a computer either at school or at home, leaving only ten percent that currently had no computer access.

In addition to possibly learning the WIMP model, source code editors present their own challenges. Editors vary greatly from environment to environment, each with their own feature set. Novice users may be confused by any (or all) of the following *commonly seen features*:

- syntax color highlighting - highlighting keywords of the programming language in a special color (or font) for easy visibility,
- auto indenting - automatic insertion of tabs or whitespace to maintain the current indentation,
- auto code-completion (assist) - displaying a list of words or phrases allowing the user to select a choice of what will be typed next (generally used to automatically complete function calls or method calls in object-oriented languages), and
- matching enclosure highlighting - highlighting of matching parenthesis or curly braces for improved readability.

Each of the above features add to the visual complexity of the editor, and possibly adding to the confusion and frustration of the user, depending on their computer background. Of the seven students from the pilot usability study run in the Spring of 2001, six of the subjects either commented on, utilized, or expressed frustration of at least one of four items mentioned above. Clearly these features have an impact on the perceived complexity of a source code editor and may increase the amount of time to learn the language of the editor.

1.1.2 Learning the Language of the Programming Paradigm

By far the most critical and complex abstract concept to comprehend by novice programmers is the syntax and semantics of the actual programming language. Sleeman [73] defines the challenges of teaching and learning computer programming as follows:

1. “*specify a detailed plan which can be carried out*” (by the programmer) - this stage involves decomposing the task in to a series of steps which can be followed to achieve the desired results
2. “*map this plan into the constructs of the target programming language*” - here the programmer must have a clear idea of the plan and what constructs are available
3. “*debugging*” (the result) - involves the coordination of information from a variety of sources that brought us to this point (such as the error messages, the program plan, the program specification and the actual source code)

⁵or Pull-down menu

From the pilot study run in the Spring of 2001, this author found that several of the subjects had the most difficulty with the second challenge. The ability to map an abstract plan into an programming language comes as quite a challenge to novice users, it is unlike anything else they may have experienced before.

Additionally, (non-trivial) programming languages contain a large assortment of syntax and semantic rules which a student must learn on their way to mastering that programming language. For example, the C programming language defines the following keywords [24]:

Defined keywords in C					
auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void
volatile	while				

C++ accepts those keywords from C, as well adding the following [24]:

Additional keywords defined in C++					
asm	bool	catch	class	const_cast	delete
dynamic_cast	explicit	false	friend	inline	mutable
namespace	new	operator	private	protected	public
reinterpret_cast	static_cast	template	this	throw	true
try	typeid	typename	using	virtual	wchar_t

Clearly with additional keywords and syntax being added, the programming language itself becomes more complex, particularly to a novice.

1.1.3 Learning the Language of the Programming Environment

Once a student has completed their initial source code entry into an editor, there are usually multiple steps that must be taken to attempt programming compilation and execution. Depending on the environment, this sequence of actions may be somewhere in the range of very obvious to very perplexing.

Microsoft Visual C++ contains a plethora of capabilities for professional-model code development related to compilation and execution, but are likely less than desirable for novice learners. In Visual C++, a user can attempt code compilation on a single file or a project (group of files). Following a compilation of one or more files, the user can *build* a project's "executable file". Building an "executable" merely involves compiling any uncompiled code and linking the resulting object files together to create an executable object.

In the usability study run in the Spring of 2001, several students exhibited confusion between the *compilation* option and the *build* option once their source code was entered. In fact, it was noted that several students always reverted to attempting a full build of the "executable" (involving compiling any modified files, linking the objects, and creating the executable file) rather than attempting only compilation on the source file with which they were working.

Several subjects even discovered that if they attempted to run the program, the programming environment would attempt to compile, link, and build the executable prior to program launch (if any of those steps needed to be completed first). In determining if the subjects did know the difference between *compiling* a source code file and *building* the executable, the following responses were obtained:

“In the beginning I didn’t have a clue as to the difference, I always used the Build command. I thought the “Linking” had something to do with it getting information from the Net, so I used Build before each run of the exe.file even when I had only changed inputfile.

It was by pure coincident during project 4 or 5 that I happened to use the Compile command and discovered it could run without having a specified input or output file, and that you by compiling basically only went over the syntax of the program...” – comment from subject #1

“I guess I realized the difference between the two (compiling source code and building an executable file) around or during project 4. I think that before then, I wasn’t really sure what was going on, I just knew which buttons to push and in what order. Now I think I know the difference, especially because of the different types of errors I get too.” – comment from subject #2

Learning the language of programming environment to achieve code compilation and execution is the third hurdle that novice students need to comprehend and master. Clearly these concepts can (and should) be abstracted to some degree for the novice user until such time that they comprehend the compile-link-execute paradigm.

1.2 Separation Between the Environment and the Compiler

Consideration must be given to the examination of the effect that both the environment (user interface and corresponding tools) and the language/compiler have upon the student. In truth, the environment is separate and distinct from the programming language and can be simply represented by Figure 1.3.

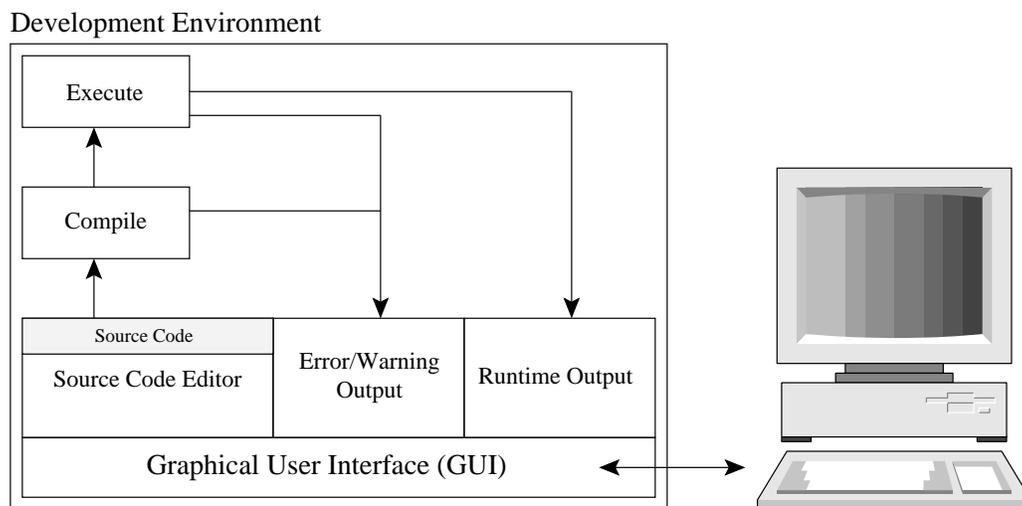


Figure 1.3: A view of traditional GUI-based programming environments

In the traditional program development environment, a Graphical User Interface (GUI) acts as a visual/virtual conduit for user actions to the compiler. Generally speaking, the GUI provides an area for source code development (typing/input) as well as a separate area for the output of warning and error messages from the compiler.

The interface, when instructed, passes the source code to the compiler where it is analyzed for correctness and translated to an executable entity. If output messages are created by the compiler, they are passed back to the environment for display to the user. Users then (attempt to) analyze the error messages (and in some cases interact with them) in an effort to correct the syntactic and/or semantic problems.

While the interface itself does play a role in the student's understanding of the editing / programming / compilation / error correction / execution process, the interface is basically limited to the management of information to/from the compiler on behalf of the user.

Some development environments allow the user to interact with compiler-generated error messages by double-clicking the mouse on them. Through this mechanism these environments will relate the error to the appropriate location in the source code and modify the view of the source code window/viewer to reflect the location of the error (thereby assisting the student in correcting the error). Additionally, some programming environments permit similar interaction with error and warning messages which occasionally lead the user to an assistive help-system textual content (in an effort to assist the user in learning the meaning of the error message). Because of the structure of this paradigm, one can conclude that the interface provided by the programming environment plays a minor to non-existent role in the comprehension of the student with respect to the syntactic and semantic errors in the development of their code.

Thus, concentration on the language compiler seems prudent in this study. Compilers play a pivotal role in program implementation and are the primary source of feedback that a student obtains while implementing their programs. Error messages produced by compilers which assume that the user has full breadth knowledge of the language and its syntactic idiosyncrasies are likely to produce error and warning messages that have no real meaning to novice programmers.

For example, in a recent draft manuscript [37] for a proposed textbook, there exists a code example of re-binding a constructor for a specific type in the ML programming language. The authors show an example of an ML error message that warns the user of the problem. The message is of the form:

```
uncaught exception nonexhaustive binding failure
raised at: stdIn:1.1-25.8
```

While the authors of the text do not necessarily require the reader to comprehend the error message, it is clear that this message is quite non-intuitive for novice programmers. Error and warning messages indicate what errors programmers have in the syntactic and semantic interpretation of their own source code. Programmers generally learn from the messages that compilers and interpreters provide. Thus error messages (like the one shown) should strive to provide meaningful insight to the programmer, no matter the level of their expertise. This notion has been long expressed, including in a passage from a 1973 programming text by Niklaus Wirth:

“...Secondly, the compiler system must produce intelligible replies under all circumstances. Particularly when used by novices, these replies will seldom be the expected computation result; rather, they are likely to be information about detected mistakes. The system must formulate these messages either in a natural language or in the programming notation being used. Never should obscure or unmotivated operating-system messages or – even worse – octal or hexadecimal dumps appear.” [90]

Given the common pedagogical model of subsetting the language in the classroom, it seems natural to extend that model to the programming environment that the students will be using, yet it is rarely done⁶.

⁶The DrScheme programming environment (see Section 2.3.1) is the only such environment using subsets still in use.

Programming environments that capitalize on such subsetting can take advantage of knowing that not only does the student have an incomplete view of the entire language, but the compiler can also know what constructs the user (student) is permitted to use.

Constructs which are inaccessible or disabled can therefore be ruled out of the student's knowledge base. With such knowledge the compiler can therefore provide error and warning messages that have less ambiguity and are more targeted to the student's current skill level.

1.3 Justification of Dissertation Topic

1.3.1 Identification of Problem

Despite the proliferation of programming environments, both commercial and those designed to support the CS1 curriculum, there is a lack of quantitative results examining the effect of these environments upon the students. The areas under examination are identified to be two-fold: the effect of the implementation of a simplified interface for the novice student's programming environment, and the effect of the enforcement of language subsets within the students' programming environment.

This dissertation examines these effects (see research questions below) by examining the implementation of a prototype environment (see proposed solution below) when introduced into the curriculum of a large number of students in an introductory (CS1) programming course.

1.3.2 Research Questions

This dissertation is based upon the following two research questions.

1. Do students respond more favorably to a software development environment which is comprised of a less complex user-interface than a development environment designed for professional programmers?
2. By applying language subsets to a programming environment / compiler in conjunction with a similar lesson plan, would quantitative academic gains (e.g. fewer compilations, fewer compilation errors, less time-on-task) be realized?

These effectively reduce to questions of the subjective impact (first question) of simplified environments, and amount of effort (second question) on the part of the novice students under examination by this dissertation.

1.3.3 Proposed Solution

In order to explore the research questions stated in Section 1.3.2, the following solution was proposed.

First, a prototype programming environment would be developed which concentrated on supporting the ideas surrounding the research questions. This includes the following two central motivations.

1. The environment (dubbed the *CS1 Sandbox*) would consist of an interface which was designed to support a pedagogical mission, not a professional one. A main tenet of this goal would be to present the interface with reduced complexity. For example, the prototype would contain fewer buttons,

menus, and text areas (GUI components) that compared to the professional strength environments (specifically compared to Microsoft Visual C++/.NET) which is traditionally used by the department's novice programmers. In addition, supporting tools such as GUI authoring aids, debuggers, profilers, source code management (versioning), and release packaging support would be removed from the environment.

Project management features such as project organization and build tools would not be implemented as it tends to be a highly complex and cumbersome abstraction for novice programmers (the prototype would be an interpreter). The prototype would support only a single source code file at one time (viewing, editing, and compilation of multiple source code files are not a part of the introductory curriculum). Finally, the environment would strive to produce error messages that were consistent with the level of proficiency of the intended users (novice programmers).

2. The environment would be implemented in two versions - one full-functionality version and the other supporting the configuration of language subsets. The first version would permit users to program using all language statements implemented in the environment, while the second version would permit use (checked at compile time) of only a subset of the full language as dictated by a configuration directives provided through the course instructor.

Second, the environment would be used in a novice programming course in order to collect data on how the implemented ideas would affect the students who would use it. The environment would be inserted into the curriculum in the CS1044 course and the student participating would be divided into three groups: those using the existing standard (.NET), the *CS1 Sandbox* full-functionality version, and the *CS1 Sandbox* version that supported the enforcement of subsets. The full experiment design is elaborated in Chapter 4.

This page intentionally left blank.

Chapter 2

A Review of Previous Work

This chapter of the dissertation looks at previous efforts of subsetting as related to both programming languages and programming environments. In addition, a review of the most recent and widely used programming environments is included when programming environments are examined. Many of these environments are tailored specifically toward novice programmers and thus they have influenced the development of the *CS1 Sandbox* discussed in Chapter 3.

2.1 Previous Efforts in Subsetting Programming Languages

Several instances of subsetting of a programming language have occurred throughout the history of programming languages. As early as 1969, Jean Sammet had considered the motivations and implications of subsetting in early high level programming languages.

“The primary motivations for subsetting are cost and time. Subsets permit smaller compilers, which can be developed more cheaply and/or more rapidly. Furthermore, subsets tend to compile faster.” [69].

Sammet continues *“...a subset of a complex programming language may be, and probably will be, very much easier than learning a subset of a complex assembly language. ...the user who does not wish to learn (or have) all the power available to him need not be bothered with the full language.”*

This section examines programming languages which implemented subsets, or those where subsetting was a related concern. Where possible, the motivations behind the subsetting efforts are discussed.

2.1.1 COBOL

COBOL [69, 72, 86] is included in this list of languages that applied subsets for two reasons: divisions and official subsets. *“The most striking feature of the organization of a COBOL program is the definitive separation of four aspects of a program into four separate and clearly defined divisions of the language.”* [69, p. 347]

Each of the divisions must appear in every program, and are used to define characteristics or components of the program. The “identification” division is used to cite the program’s author, name, date, and general

comments. The “environment” division is used to list the hardware on which the program is expected to be compiled and executed. Descriptions of the data records and input files are placed in the “data” division, while program statements are added to the “program” division. In addition to readability, the four divisions in COBOL were expected to aid in achieving compatibility across computers present during this time.

In addition to divisions, COBOL was noted for a set of official subsets that were considered optional to compiler developers, depending on the target platform.

“Although at the start of the Short Range Committee’s activity it was tacitly assumed that COBOL was being developed only for large computers, an increasing number of manufactures and users became interested in having this available on smaller machines. As a result, it was necessary to try to provide some official subsets that would be an adequate subsection of the entire language but still be more easily implemented on small machines than the entire language would be. In the COBOL-60 specifications, a subset called Basic COBOL was defined. In the 1961 specifications, a subset entitled Required COBOL-1961 was defined to consist of “that group of features and options, within the complete COBOL specifications for the year 1961, which have been designated as comprising the minimum subset of the total language which must be implemented (to the extend of hardware compatibility) by any implementor claiming a ‘proper’ COBOL-1961 compiler”. All other features and options were considered elective; but if they were implemented, they had to be done so in accordance with the specifications given in the manual. The manual for COBOL-1961 Extended kept this idea, but the concept was later dropped entirely. A subset known as Compact COBOL was defined by a COBOL Committee subcommittee but was not published because of possible confusions with the standards work.” [69, p. 339]

2.1.2 BASIC

The BASIC [87] programming language was designed as a programming language for non-science students¹ at Dartmouth College in the early 1960s by John Kemeny and Thomas Kurtz. Their goal was a simple one – “*devise a computer system that would be friendly, easy to learn and use, and not require students to go out of their way.*”

In the early days of defining what would become the BASIC programming language, it’s authors disagreed on its foundation.

*“While Kemeny instantly realized (in the spring of 1963) the need for designing a new language, I preferred to believe that, although existing languages such as FORTRAN and ALGOL were not suitable, reasonable subsets of these languages could be devised. This belief proved wrong. For example, the compound statement is a central concept in ALGOL; one cannot construct a nontrivial **for-statement** without it. But the idea of a compound statement posed pedagogical problems; it would have to be defined and explained....With FORTRAN, a subset wouldn’t make sense if it violated the IJKLMN convention....Very early then I agreed with Kemeny that a new language was needed to meet our requirements.” [87, p. 519]*

Despite creating a new programming language for their students, BASIC did borrow from FORTRAN and ALGOL – both in use previously. BASIC borrows the three loop-controlling values (initial, final, and step) from FORTRAN, as well as reuses two keywords from ALGOL – **for** and **step**.

¹At the time, only a quarter of the students were majoring in science or engineering. It is apparent that sufficiently large numbers of non-science students were enrolled in computing courses to warrant addressing the need for a pedagogical programming language.

2.1.3 FORTRAN / Java

FORTRAN and Java are included in this list of languages related to subsets because of a simple reason – releases. Since their initial releases, both languages have undergone numerous revisions and re-releases where additional functionality has been added to the language over time. The same can be said for many other programming languages.

Since its inception in 1955, FORTRAN has been superseded by the release of FORTRAN II, FORTRAN III, FORTRAN IV, FORTRAN 77, and FORTRAN 90. Along the way, the language picked up such items as:

- the `if-then-else` and `while-do` statements (FORTRAN 77),
- “free format lexical conventions...user-defined data types, modularization mechanisms, and pointers” [46] (FORTRAN 90), and
- “exception handling, parameterized types, and object-oriented programming” [46] (FORTRAN 95).

Java has undergone similar changes since its initial release in 1995. Along the way, Java has added enhanced graphic user interface capabilities, support for XML, security, additional input/output streams, 3D graphics, and support for internationalization just to name a few. In the forthcoming release, the language is expected to add enumerated types, generic types, and `for-each` loops.

In both cases (and additionally with other languages that have undergone iterative releases), one could argue that earlier releases of a language generally become subsets of later releases. This is not always the case, as later releases occasionally remove components from the core language (best evidenced by the deprecation of methods and objects in Java).

2.1.4 Ada

Ada is particularly noted for the strict avoidance of subsets [4, 88]. The rationale behind the development of Ada was to decrease software development costs by increasing reuse and portability. These goals were to be achieved by the mandating of one common Ada language to be used in all DoD projects. If there existed no Ada subsets, then portability of an application would be more likely across projects (or at least code-reuse seemed to be more easily supported) as there would only be one compiler for the language.

“Therefore, DoD [Department of Defense] took the unprecedented action of registering the name “Ada” as a trademark. This provides the ability to control the use of this name and to guarantee that anything called “Ada” is the standard language. That is, subsets and supersets of Ada cannot legally be called “Ada”. [47]

The Department of Defense took this no-subset stance following the release of Janus/Ada in 1981 by RR Software, Inc. [43, 68]. Janus/Ada was released for the personal computer and was a subset of the Ada language. RR Software has continued to release two additional versions of their Ada compiler for the personal computer – Janus/Ada 83 and Janus/Ada 95.

The SmallAda programming environment [26] applied a subset of the Ada programming language (the “Pascal subset”) with the multitasking support of the Ada language. This environment was not developed for novice programmers and thus was not examined more closely.

2.1.5 Microsoft .NET Common Language Specification

Microsoft's .NET is a recently deployed application development framework for their next generation of application development across the Microsoft platform(s). Within the .NET framework are several tools for application development (programming environments) supporting a number of programming languages, a runtime platform for application execution, and other tools used in the deployment of applications under the .NET paradigm (such as application servers).

One component of the .NET framework in particular is the Common Language Specification (CLS) [54] which is used to define on which features and conventions that all languages under the .NET banner will be able to rely. Since applications developed using .NET can include modules developed in different languages, this specification defines a consistent set of rules common to all of the .NET languages so that they may inter-operate.

Within the CLS there exists the definition of a subset of the common type system [51] which is used to specify *“how types are declared, used, and managed in the runtime”* [51]. For each application module authored by a .NET programmer, the source code can contain an instruction to the compiler to ensure that the source code is CLS compliant (warnings are issued in the event that it is not). By following this process of including a compiler directive into the source code, a programmer can restrict themselves to the type subset enforced by the Common Language Specification in .NET.

2.2 Older Pedagogic Programming Environments (Pre-1995)

Since 1995, a greater emphasis has been placed on the creation of programming environments specifically designed for novice programmers. This focus seems to have been brought about with the release of the Java programming language and its subsequent use as a first year programming language. Prior to 1995, an eclectic group of environments existed for similar purposes, but their release seemed to be less frequent than those seen today.

In the subsections that follow several programming environments and tools which were designed for the novice academic programmer are described. Their efforts explored ideas still being implemented in programming environments for novice users today.

2.2.1 The Cornell Program Synthesizer

The Cornell Program Synthesizer [77] was an early (1981) program development environment developed at Cornell University. The system was designed to support introductory programming courses at Cornell teaching PL/CS, a derivative of PL/I. Additional universities adopted the Synthesizer for their courses, including Rutgers University and Princeton University.

The Synthesizer's goal

“...was to develop a unified programming environment that stimulates program conception at a high level of abstraction, promotes step-wise refinement, spares the user from mundane and frustrating syntactic details while editing programs, and provides extensive diagnostic facilities during program execution.” [77]

To achieve these goals, the Synthesizer provided a collection of immutable grammar templates which were placed together by the programmer to create a program.

The templates provide a structured method to “...view that a program is a hierarchical composition of syntactic objects, rather than a sequence of characters.” Essentially, the programmer selects a template to place in the program at a desired location. Then, using the keyboard controls, the user navigates a syntax-aware editing cursor in order to correctly place specific expressions or statements to complete the template.

For example, a conditional statement template is of the form:

```
IF (condition)
  THEN statement
  ELSE statement
```

Once inserted, a user can then complete the condition expression or embed an additional statement template in the THEN or ELSE lines at the appropriate locations.

The PL/CS templates provide subset-like behavior to the end user in that no template can be applied to a part of the program which makes then makes the program syntactically invalid. Thus, only a subset of the complete list of templates may be used by a programmer at any given time, depending on the location within the program. As templates are inserted, or additional expressions/statements are added to the template (known as phrases), the modified code is validated to ensure correctness.

By focusing the user on the insertion and manipulation the templates, the Synthesizer environment clearly promotes a higher view of the design and creation of a program. The syntax-aware editing cursor supplements the templates nicely by permitting the user to only navigate to and edit those portions of a template which can contain phrases or nested templates. In doing so, the environment assists in the correct syntactic assembly of a program.

Attempts to obtain a working copy of the Synthesizer for evaluation and screen capture generation were unsuccessful.

2.2.2 Pascal Trainer

Webber’s Pascal Trainer [85] was a pedagogical sandbox for students working in Pascal at Western Illinois University. The Trainer was a DOS-based interpreter (see Figures 2.1 and 2.2) which permitted students to submit simple expressions, statements, functions, and procedures. Western Illinois used the Pascal Trainer in a series of closed-laboratories where “students solve a carefully planned sequence of well over a hundred simple programming exercises”.

The Trainer’s goal was to “create a system for learning programming that would build logically from simple expressions, though statements, to functions and procedures”. This goal, not stated as such, clearly supports the notion of implementing an environment which students would use to work through the subsets of a programming language. Webber likens the process to that of learning a natural language, where novices first learn simple utterances and gradually build to more complex statements.

Commenting on then-modern development systems, Webber points out that:

“When the students have to use a development system right away for assignments, the teacher has no choice but to present language topics in a preposterous order. Turbo Pascal doesn’t recognize anything smaller than a program, so you must immediately teach how to write complete programs. The only way to tell if a program is working is for it to produce output, so you must immediately teach I/O statements. Any full language system has a complicated interface, and you must teach this interface right away.”

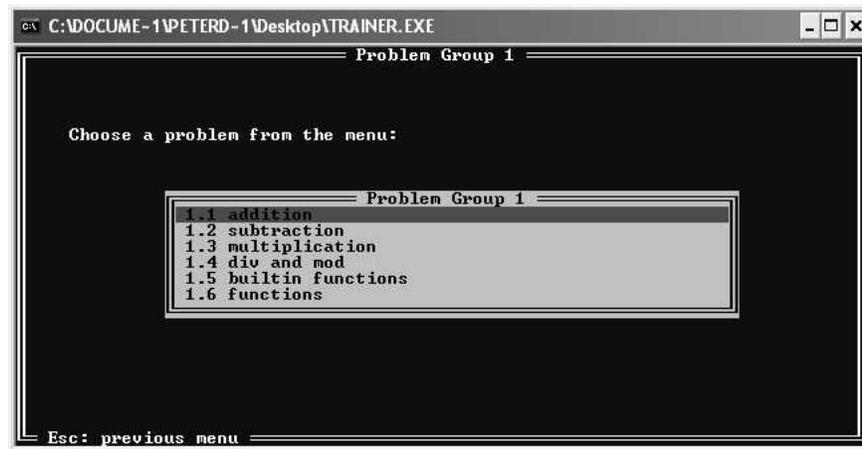


Figure 2.1: The Pascal Trainer's interface being used to select a module of exercises

The Pascal Trainer was placed in this category of environments (Pre-1995) for two principal reasons. First, development on the Trainer began prior to 1995 and secondly, its DOS-based menu interface separates it from the more modern group of environments developed after 1994. The Pascal Trainer was not used following 1996.

2.2.3 Emile

Emile [35, 36] is a software programming environment for the creation and simulation of physical models which address kinematics (physics concept of addressing motion without concern for forces). Emile was developed by Mark Guzdial of the Georgia Institute of Technology to implement *software-realized scaffolding* and was evaluated during a three week summer session that involved five high school students.

Scaffolding, one of Guzdial's major research interests, is the term used to define supporting techniques to assist a student who is performing an activity with a goal of learn a particular skill . In Emile, the software scaffolding supports the creation and simulation of kinematic physical models by the students while learning concepts in physics.

Guzdial identifies three key types of support from prior works that are combined to provide scaffolding. He attempts to incorporate aspects of each into the Emile environment. These support types include the following:

- Communicating process - The process being studied is communicated to the student in a manner that includes the highlighting of key points. Guzdial points out the when the process that is being studied is structured (simplification can be a form structure), it is easier to communicate to the student.
- Coaching - A coaching presence provides feedback, encouragement, corrections, comments, etc. to the student. A balance of positive feedback and learning through failure are also mentioned as key issues with coaching.
- Eliciting articulation - Articulation on the part of the student encourages reflection on the process being learned. Generally, this comes at the initiation of the source of the scaffolding. Guzdial likens this to a master-apprentice relationship, where the master quizzes the apprentice.

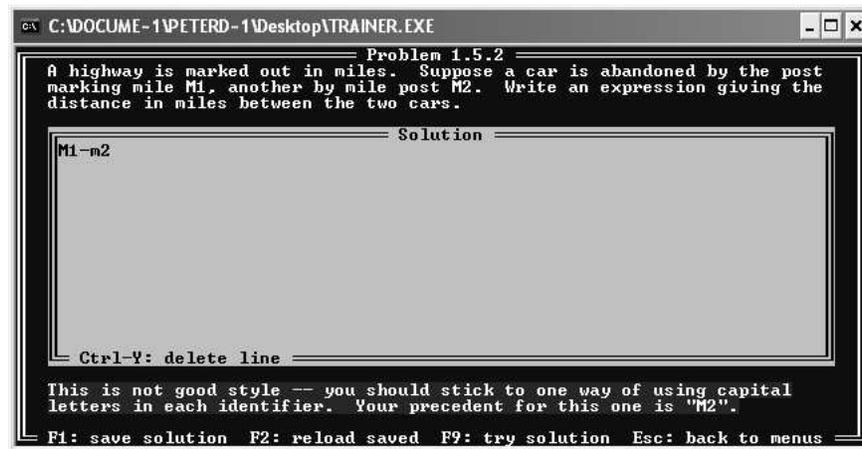


Figure 2.2: The Pascal Trainer providing an exercise for a student

An additional notion that Guzdial stresses for scaffolding [36] is the notion of *fading*. Fading is the gradual reduction of the scaffolding in the learning process. As a student becomes more proficient in the skill they are studying, the scaffolding should become less prevalent. Scaffolding can fade as the result of user initiated actions (user selection) or in a more automated fashion (system analyzes the needs of the student and fades the scaffolding automatically).

Guzdial asserts that students have three critical problems that hinder the learning of programming languages:

- Assembling programs is hard - learning to understand the different possible combinations of a programming language's components is a complex process. Achieving programming goals by combining programs is challenging.
- Syntax is complex - The combining syntactic elements can lead to dealing with syntax problems while attempting to deal with semantic problems as well.
- Students lack an understanding of computational process - Students do not have a clear understanding of programming language interpretation and execution. Clearing up the students' understanding of these issues can lead to a better understanding of their own programs, yielding more successful programming results.

2.2.4 Solo

Solo [25] is both a language and programming environment specifically designed to help instruct novice students the concepts of programming using a database manipulation paradigm. Solo was developed in the late 1970s in support of a Cognitive Psychology course at The Open University by Marc Eisenstadt. With the increasing content of Artificial Intelligence (AI) being added to the course at the time, students were required to complete a programming project as part of the course. The project had five goals:

1. establish a connection to a computer (connect and log in),

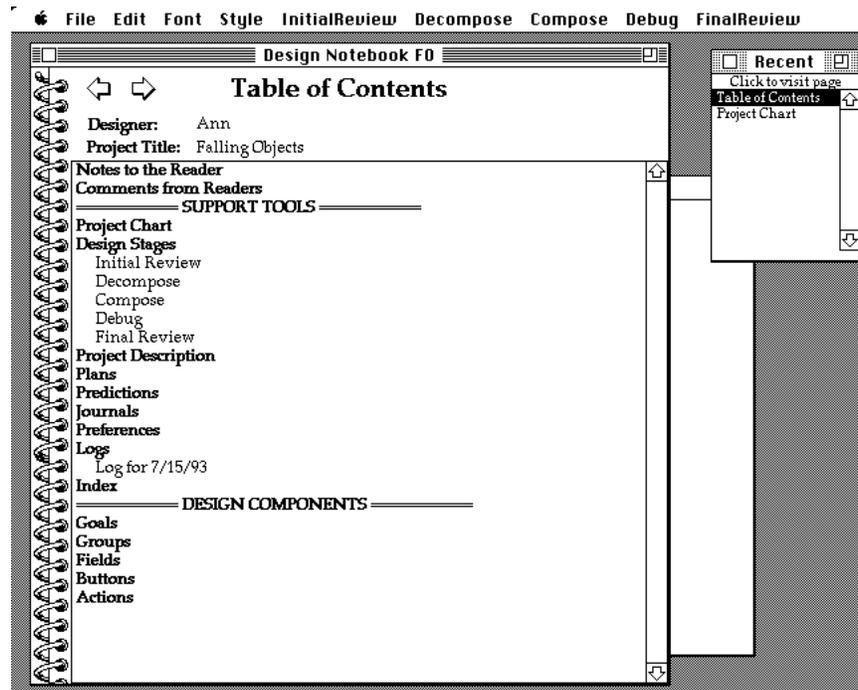


Figure 2.3: A view of a new, empty project in the Emile environment. Shown in the image is the student’s Design Notebook, used to contain “...all of a student’s programming representations (such as the node-and-link Project Chart view of their project) all of a student’s articulations about the project (such as Journal entries and Plans) and descriptions of all the components of the project - each on a separate page of the Notebook.” [35]

2. be able to define each value in a list of computing terms (symbol, computer, data base, variable, etc.), and be capable of explaining the relevance of each to the models of cognitive psychology,
3. write complete programs using each of the computing terms,
4. provide explanations of the components of the project which embody aspects of cognitive processing, and
5. to explain why “representation” is a problem in computer programs.

Due to the unique structure of The Open University², Eisenstadt and his team had to decide how best to support the goals of the project.

“We therefore had to precede with great caution – we had to produce a nonthreatening computer programming experience for students with no prior knowledge of computing who would be working alone at remote student centers and who were not interested in computing in the first place.”

²The Open University has no entrance requirements, and the Cognitive Psychology course at the time required no prior programming experience.

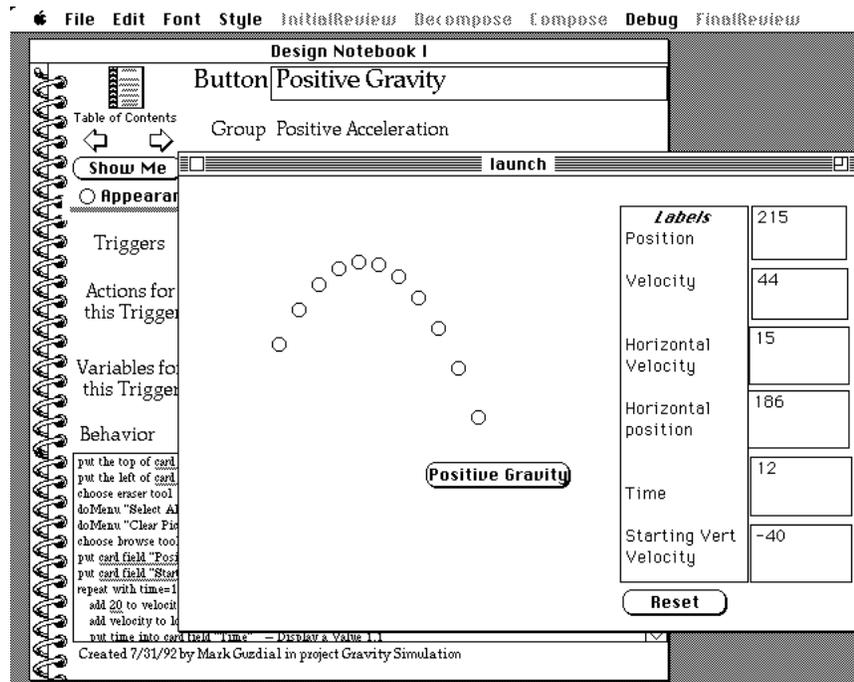


Figure 2.4: A project in Emile which simulates two-dimensional projectile motion

What emerged from the effort was a text-based programming interface and database manipulation language. Simple procedures invoke a small set of primitives to add, remove, list, and print values from a database structure. The structure itself is a “unidirectional labeled graph with arbitrarily named nodes connected by arbitrarily named relations” (see Figure 2.5).

```
PETER
|
+--ISA-->MAN
|
+--HAS-->BLACK_HAIR
|
+--LIKES-->SODA
```

Figure 2.5: An example of a data structure in the Solo language

The Solo language is comprised of only ten primitives which permit the manipulation and querying of the database, as well as supporting output, procedure definition, program source code editing, iteration, and session termination.

In an effort to provide a friendly interface for novice students, Solo provided several key features. These included the following.

- Automatic line numbering (in a BASIC-like notation).

```

TO ASSESS /X/
10 CHECK /X/---VOTES--->INDEPENDENT
    10A If present: PRAISE /X/; CONTINUE
    10B If absent: CONDEMN /X/; CONTINUE
20 PRINT "NO MORE TO SAY ABOUT" /X/
DONE

SOLO: ASSESS MARY
Oops...I don't have a procedure called "CONDEMN"

SOLO: HELP
You've tried to use a procedure without first having defined it.
See pp. 37-39 of Units 3-4 for how to do this, or else type
    HELP PROCEDURES
By the way, this error may simply have been caused by
a spelling mistake!

```

Figure 2.6: A sample help sequence in the Solo programming environment. (Note that the definition of the PRAISE procedure is not shown, and the CONDEMN procedure definition has not yet been provided – prompting the error messages(s).)

- User-defined procedures were limited to ten statements each.
- The GOTO command was prohibited.
- Solo primitives have an English-like mnemonic form. They are designed to closely correspond to the job the primitive is to carry out. For example, the language provides the primitives NOTE (adds an information triple to the database), FORGET (removes an information triple from the database), and DESCRIBE (used to print a database structure).
- Obvious spelling errors (near-misses on procedure names) were automatically corrected by Solo, while misspelling node or relation names generated a prompt for the user to confirm the suggested correction.
- User-friendly help was provided on demand which either expanded on an immediately preceding error message, detailed the use of a primitive, or provided an index of available help. The sequence in Figure 2.6 demonstrates the idea behind Solo's support.
- Students could modify the Solo programming environment by self-enabling/disabling environment functionality such as spelling corrections, procedure tracing, and the displaying of changes to the inner database following modification command execution. By doing so, students could have the environment grow with them (become increasingly more complex and provide additional support tools) as they became increasingly comfortable with the environment.

The Solo language and programming environment provided a platform to support a Cognitive Psychology course for over twenty-five hundred students in up to two hundred and fifty study centers throughout the United Kingdom. The language was used to introduce novice programmers to the notion of “assertational database” and “pattern-matching” by completing programming projects in the Cognitive Psychology course. Students were reported to find these concepts “highly intuitive, motivating, and relevant to their academic backgrounds”.

2.2.5 Bittitalk and View Matcher

Carroll and Rosson’s work at IBM’s T.J. Watson Research Center on the Smalltalk programming environment [15, 32, 65, 66, 67] has led to several interesting works related to this dissertation. Rosson indicates [65] that Smalltalk is different than most conventional languages.

“Our experience with object-oriented programming, specifically with Smalltalk, confirms that while beginners are frequently intrigued by the language and its environment, they are also often overwhelmed....Smalltalk is an environment, not just a language; it provides a number of very sophisticated tools for accessing and manipulating code. Gone is the old-fashioned paradigm of writing code in a text editor, compiling and executing. Instead, users are introduced to hierarchy browsers, workspaces, inspectors, debuggers, a system transcript and a disk browser. These tools are critical aspects of the power of Smalltalk for an experienced programmer, but they are further overhead for the beginning Smalltalk programmer.”

This unique environment provided the catalyst for intentionally subsetting the environment and language – Smalltalk contains a vast number of object-oriented classes³. From this Smalltalk environment grew the Bittitalk browser and View Matcher filter views.

The Bittitalk browser is a “filtered view” [66] of the traditional Smalltalk programming environment which provided visibility to only a selected subset of classes and methods in the Smalltalk language. The browser provides access to less than 200 methods selected by the browser developers to be the most appropriate for novice Smalltalk programmers. Bittitalk’s goal, which closely resembles the goal of this dissertation, is to “shield new users from parts of the class hierarchy” in an effort to thereby reduce the learning curve for new programmers.

However, as the authors point out [65], initially learning the language is only one hurdle to becoming an experienced Smalltalk programmer. Bittitalk filters existing classes and methods for novice users, but does nothing to help more experienced users track the internal dynamics of an executing Smalltalk program.

The Smalltalk View Matcher browser takes a similar approach in filtering the view presented to programmers, however it does so from within an executing Smalltalk application. Object-oriented programming languages utilize the passing of messages between objects to accomplish interactions within executing software. These interactions encompass everything from expression calculation, to user-interaction events, to updating the graphical user interface. Due to the large number of messages passing between objects in Smalltalk programs, the View Matcher was created to assist in allowing the programmer to pause the execution of an application and examine the objects and methods currently in use to better understand the structure of the application, the interaction among objects (via message passing), and the objects themselves.

The View Matcher took on a two-pronged approach in its implementation. The first prong concentrated on providing such filtered views (graphically⁴) of the composition of objects and message passing capabilities via instructional application samples for novices. Specific applications (e.g. a blackjack game) were implemented and used with the View Matcher as instructional aids. The aids were based upon prior user interaction analysis attempting to support three fundamental user concerns during the learning of Smalltalk: “What Can I Do?”, “How Does It Work?”, and “How Do I Do This?”.

The second prong of the View Matcher implementation also concentrated on presenting object composition views, however they were tailored to support the actions of experienced programmers who wished to reuse

³[65] states that Smalltalk (at the time) contained more than 2000 methods available for programmer use. Because of its size, it was sometimes known as the “encyclopedia” [44].

⁴Refer to the references for screen captures of the browsers in use.

existing classes and objects in the large library provided under Smalltalk. The View Matcher attempts to “...provide just the information a programmer needs to incorporate a component into an ongoing design project.”

Complete analysis of the impact of the browsers on IBM programmers was not provided in the literature regarding the Bittitalk Browser and the View Matcher filters. However, anecdotal results [66] seem to indicate that learners’ understanding of the Smalltalk environment and classes were improved after only a few hours of use with the browser and filters.

2.2.6 Karel the Robot

Karel the Robot [61] is a groundbreaking programming language released by Richard Pattis in 1981 for the purpose of providing a framework for the introduction and instruction of programming concepts to novice students. Karel presented the student with a robot (named Karel) which could move about a two dimensional grid and perform a variety of simple tasks (location sensing, object manipulation, etc.) Using the special Pascal-like language developed to support the robot, students could author programs to direct Karel’s movements and activities.

Karel’s world (the two dimensional grid) is comprised of intersecting streets and avenues which run east-west (streets) and north-south (avenues). Karel has the ability to turn in place to face toward one of the four cardinal compass points (north, east, south, west), as well as move one block at a time along the streets and avenues in its world. The grid world in which Karel operates is bounded by two infinitely extending walls along the western and southern edges of the world (see Figure 2.7 for a depiction of the world Karel operates in). Wall segments may also be present across an avenue or between streets in Karel’s world, so that movement is hindered. At any given street and avenue intersection one or more sound-emitting hazard cones (known as beepers) may be picked up (if present) and placed in Karel’s beeper bag. Additionally, Karel may remove beepers from its beeper bag and place them at any intersection. Karel’s cameras give it limited sight (only one-half of a block in three directions), providing the ability to detect walls in its path or near its proximity.

Karel’s language is procedural in nature, providing the capability to create simple encapsulated user-defined abstractions (functions) and includes iteration and conditional constructs for student use in manipulating the robot. The Karel system (language and robot) is used as a springboard to introduce programming to novice students. Keywords in the supporting language are highly similar to natural language commands (e.g. *move*, *turnleft*, and *pickbeeper*) in an effort to ease the acclimation process to computer programming. The visual nature of the robot environment provides instant feedback for users to validate program correctness.

The Karel environment has itself been the subject of modifications and extensions since its original release in 1981. The most notable of these extensions include Karel++ [10] and JKarelRobot [14].

Bergin’s work with Karel++ updates the language to utilize an object-oriented paradigm drawn from C++ and Java, moving movement and control statements to be methods associated with a robot class/object (e.g. *Karel.move()*, *Karel.turnleft()*, and *Karel.pickbeeper()*). Object instantiation is also supported within the Karel++ language, thereby permitting students to instantiate robots of any name. As Karel was a springboard for a migration toward a more mainstream procedural language (such as Pascal), Karel++ is used as a springboard toward learning C++ or Java. The Karel++ simulator has been developed in the Java programming language to maximize platform accessibility. A subsequent version of Bergin’s implementation, Karel J. Robot [9], is deployed as a Java library to be integrated into (called from) student programs using the full Java language (not an object-oriented pseudo-language) or utilized via Java programming environments (such as BlueJ – see Section 2.3.4).

The JKarelRobot effort created an unified platform for working with the original Karel idea under three different language paradigms – procedural, object-oriented, and functional. The JKarelRobot environment (see Figures 2.7 and 2.8) was developed using the Java programming language to assist in maximizing the potential client base using the environment.

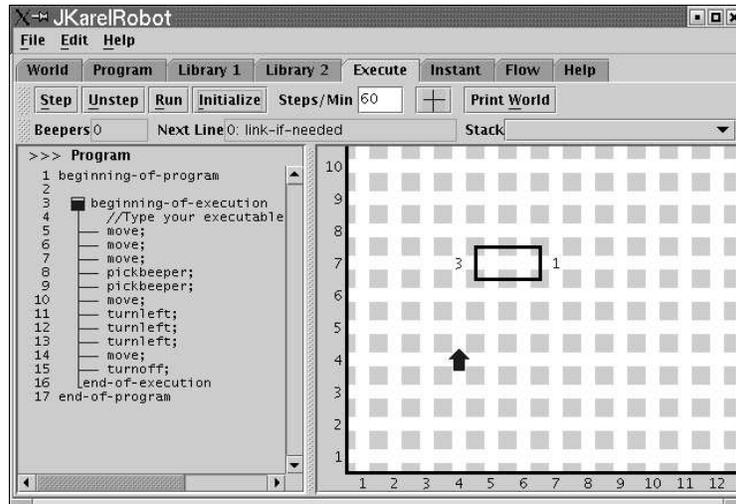


Figure 2.7: The JKarelRobot programming environment. This screen capture shows the execution of a Pascal-like program (authored in another portion of the programming environment). Execution may be performed a step-at-a-time or all-at-once by the student (without interaction).

Buck and company additionally integrated several features into their environment to further enhance use and interaction with Karel and Karel’s world.

- the Instant Window – A textual region of the interface where student can directly enter programming statements to Karel and see them interpreted immediately without the requirement for a complete program to be present. This “interpretive mode” is similar in nature to direct object-manipulation capabilities of the BlueJ programming environment (see Section 2.3.4) or the interaction pane seen in the programming environment (see Section 2.3.2).
- CSD Support – JKarelRobot supports the displaying of Control Structure Diagrams (CSDs) (as seen in the jGrasp environment in Section 2.3.3) to help “give visual cues to the dynamic behavior of the code”.
- Flow Charting support – A flow charting tool is provided permit student exercises. Buck reports that students were tasked to translate control structure diagrams into classical flow charts to assist in reinforcing “the meaning of the control structures”.

Becker [8] chose to implement Karel directly in Java, providing a application programming interface (API) which can be accessed from Java programs to represent Karel’s world, as well as control Karel. In doing so, Karel can be used directly in a curriculum which begins instruction using the Java language (rather than a pseudo-language). One strength in this approach is the ability to use the full breadth of Java in interacting with Karel, including the ability to instantiate and access a wider variety of objects “including objects which do not extend `Robot`, local and instance variables of any type and threads.”

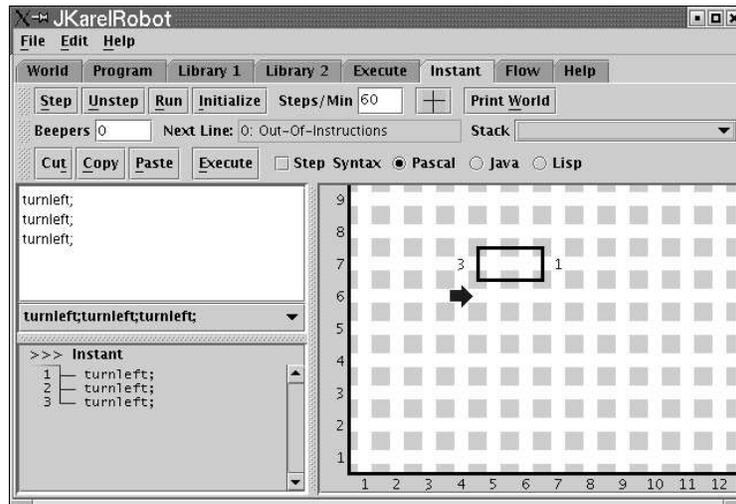


Figure 2.8: The JKarelRobot programming environment’s instant interface. This screen capture shows a student issuing command statements to the Karel robot without the need for an encompassing program.

2.3 Current Pedagogic Programming Environments (1995-2003)

As mentioned previously, since 1995, there has been an increased emphasis in the computer science education community in the development of programming environments for novice students. In the sections that follow, the most prevalent of these environments are examined.

2.3.1 DrScheme

DrScheme [27, 28, 29] is a Scheme programming environment targeted toward CS1 students who are beginning to learn how to program. DrScheme is product of the Programming Languages Team at Rice University.

DrScheme provides a graphical user interface to a Scheme programming environment (see Figure 2.9). The approach by this programming environment however, is unique. It utilizes a *tower of Scheme subsets* within the programming environment. Currently, there are only four such subsets in the DrScheme environment: *Beginning Student*, *Intermediate Student*, *Advanced Student* and *Full Scheme*, each of which are coded into the application by the DrScheme developers and cannot be modified. Users can select a language level (*Beginning Student* is the default level) prior to developing a Scheme program in the environment. The advertised design of DrScheme is that each of the levels matches the natural introduction of Scheme to novice students.

Additionally, DrScheme strives to detect syntactic mistakes and return error messages to the user appropriate to the level of the language that they are working in. For example, Scheme is a language that uses prefix notation for its mathematical expressions. If an introductory student reverts to an infix notation (which they would likely be familiar with from secondary education), they may either receive questionable results or an error message that is highly confusing. The DrScheme runtime environment in the *Beginning Student* level looks for such notations and informs the user that the notation is invalid.

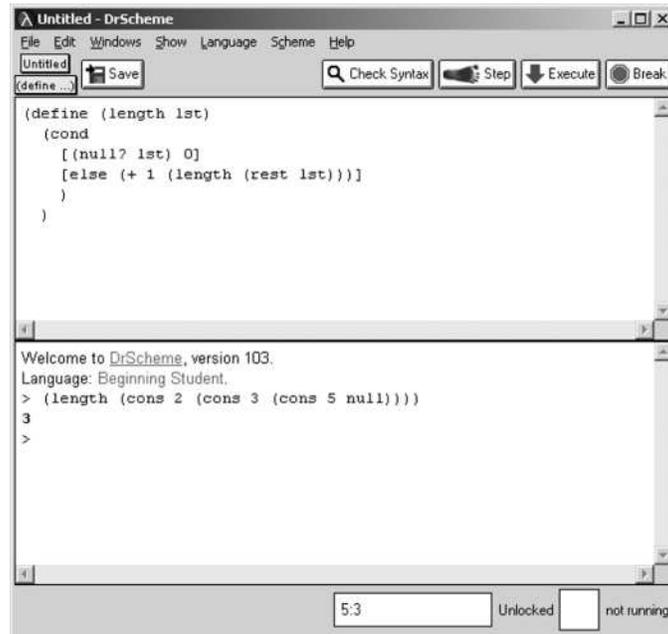


Figure 2.9: The DrScheme programming environment

For example, the following code is actually valid in full Scheme.

```
(define (length lg)
  (cond
    [(null? lg) 0]
    [else 1+(length (rest lg))]
  )
)
```

In DrScheme the `else` expression is syntactically valid but is semantically invalid as the `1+` is treated as an identifier (which is undefined) rather than an expressions' operand and operator. However, for introductory students, it is highly likely that the student did not intend to write the code in that manner and intended to write:

```
(define (length lg)
  (cond
    [(null? lg) 0]
    [else (+ 1 (length (rest lg)))]
  )
)
```

In the Beginning Student level, DrScheme mandates this limitation and responds with the following message:

```
cond: clause is not in question answer format
```

Thus, the error message is present in one level (Beginner) and disabled in the other (Full).

DrScheme also adds several other pedagogic enhancements to the programming environment. By and large, these additions are Scheme (or functional programming) specific and do not readily apply to a procedural programming environment). For example, DrScheme also provides:

- Run-time error mapping to source code - traditional programming/run-time environments for Scheme do not link the error message with the location of the error in the source code. DrScheme presents an error's location in the source editor following the error detection and termination of execution.
- Stepper - the symbolic stepper allows students to view (one step at a time) the reductions⁵ as the program executes. Students can decide which reduction steps are shown using the stepper tool.
- Annotations - DrScheme can also annotate the coding display to show (point out) syntactic elements such as primitives, keywords, bound variables, free variables and constants.

DrScheme has a particularly excellent code editing/display window. In addition to syntax color highlighting (for enhancing the display of keywords and constants), the editor fills the background of statements with a gray color when the cursor is positioned next to parenthesis and brackets. One particular frustrating experience this author has had with Lisp and Scheme environments in the past is the utterly confusing parentheses structures that can arise in the most simple of programs. The background fill immediately helps to properly visualize the matching of parenthesis while developing Scheme code.

While DrScheme is one of the most recent programming environments to utilize layers (or towers of the language as referred to in the documentation), the Programming Languages Team have not published any empirical evidence on the efficacy of improving student learning through their environment. Rather, they have taken an evangelical approach to encouraging potential instructors and students of CS1 to switch their curriculum to Scheme, citing numerous secondary school systems that have done so for their introductory programming courses.

2.3.2 DrJava

DrJava [2, 3, 74] is another programming environment for the development of Java programs. DrJava, like DrScheme (see Section 2.3.1) is product of the Programming Languages Team at Rice University. Designed specifically for introductory students, DrJava provides the following features:

1. The user's interface is comprised of three panes – the documents pane (allowing quick access to multiple open source code files), the definitions pane (where users can create and edit source code), and the interactions pane (where users can evaluate Java expressions and/or statements via a “read-eval-print loop” (REPL)). An example of the interactions pane in use can be seen in Figure 2.10 where a Money object was instantiated by the user. Following the instantiation of the Money object, several method calls were performed on the object, including an attempt to call a non-existent method.

The interaction pane is not unlike the BlueJ interface (see Section 2.3.4) for interacting in real-time with Java objects and classes via the programming environment's interface. Allen argues [3] that the REPL interface “...is often a better tool for program debugging than a conventional debugger” because

⁵Reductions are the step-by-step process of reducing expressions in programming source code. Generally these are mathematical in nature, but can include function calls or other expressions that will evaluate to some type of programmatic result.

“...it does not require them [beginning programmers] to learn the mechanics of using a debugger such as how to set and unset breakpoints, how to dump the stack, and how to query the value of variables.”

2. DrJava can be executed in conjunction with (not by default) Sun’s Java debugger that is a component of the Java Software Development Kit (SDK). The SDK is required to be installed in order to execute DrJava as well as compile source code developed with the environment. DrJava provides a simple interface for the examination of the runtime stack, breakpoints, and watches set on individual variables. The debugging interface is an integrated part of DrJava, however the underlying debugger relies on the SDK debugger.
3. DrJava can be integrated with the JUnit [31] testing framework. With the JUnit libraries installed and the appropriate programming conventions followed (JUnit has a strict set of conventions for automated testing of Java classes), the DrJava programming environment provides easy one-button access to initiating JUnit tests on the source code file that is currently being displayed.

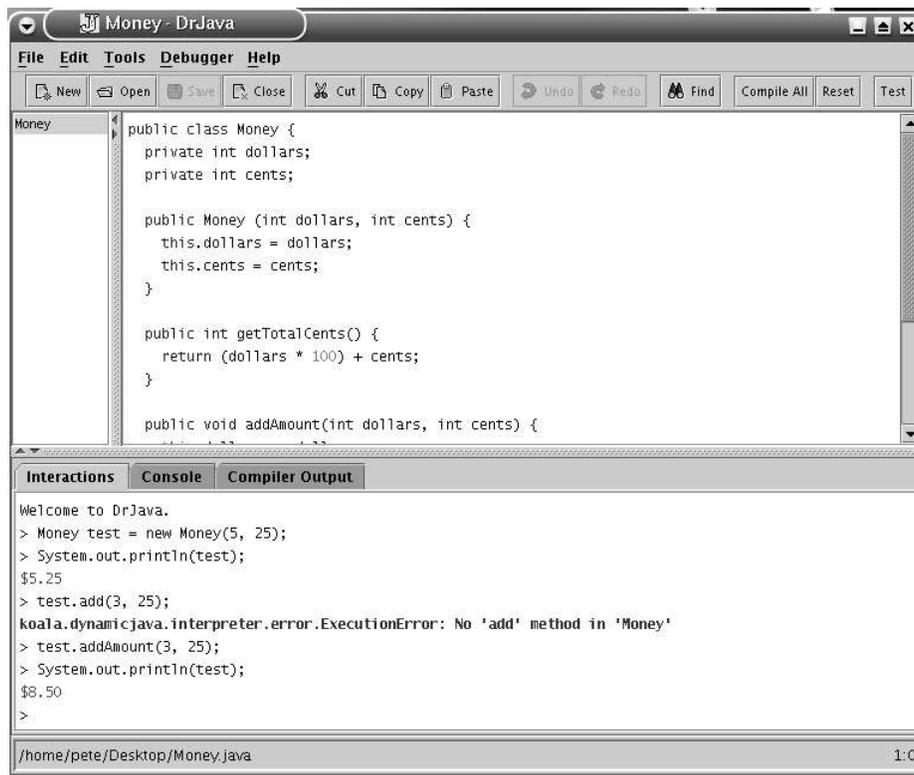


Figure 2.10: The DrJava programming environment

DrJava’s definition pane (where source code is entered) offers several user-friendly features for novice programmers, including syntax color highlighting, brace matching and automatic indentation. These features are updated following each keystroke by the user.

Despite being developed by the team from Rice University which developed DrScheme, there are no language subsets present (at this time) in the DrJava programming environment. DrJava is being

developed as an Open Source application with additional features for advanced developers are currently being planned. These features include an improved debugger, and the ability to generate and display Javadoc documentation from Java source code files.

2.3.3 jGrasp

The jGrasp [19] (Graphical Representations of Algorithms, Structures, and Processes) project at Auburn University is a software development environment that parses Java, C, C++, Ada, or VHDL (Very High Speed Integrated Circuit (VHSIC) Hardware Description Language) input, and produces control flow and data structure diagrams of the source code input. jGrasp output aims to assist in the comprehension of a program flow of control during execution. This is accomplished by overlaying the coding display (see Figure 2.11) with a series of graphical notation objects [21] (lines, boxes, ellipses) to denote the relevant portions of the input (source code) which affect the execution path.

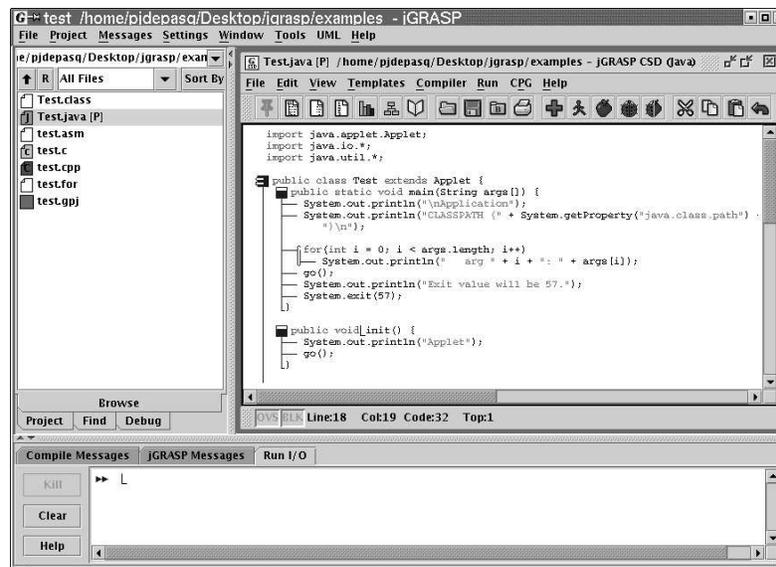


Figure 2.11: The jGrasp development/visualization environment displaying a control structure diagram (CSD).

jGrasp utilizes a pre-installed compiler to provide a programming environment in which a user can author, debug, and visualize their programs. However, the graphical notation that jGrasp provides can be generated without the use of an installed compiler. That is, the compiler is required only using the environment in which to compile the code. jGrasp is implemented with a language recognizer to support the creation of the notation.

Since first examining jGrasp in conjunction with this dissertation (Spring 2001), other additional features have been added to the environment. These include the following.

- Project management - a project management tool has been added to assign individual source code files to a project. Once a project has been created, project specific settings can be adjusted (print settings, compiler settings, etc.) as well as using the project as the basis of additional features (Javadoc and Jar file creation in the case of Java source code files, for example).

- Unified Modeling Language (UML) Class Diagrams [57] - UML diagrams can be generated for all Java source code and class files (from Jar files) contains within a project (see Figure 2.12). Users can manipulate the diagram's layout, and interact with the class, method, and field representations to directly view the corresponding source code.
- Complexity Profile Graph - The jGrasp environment can generate a complexity graph (see Figure 2.13) upon request for a given source code file. The metric "...is based upon a profile metric which is designed to compute complexity at various levels of granularity based on the underlying source language." [20]

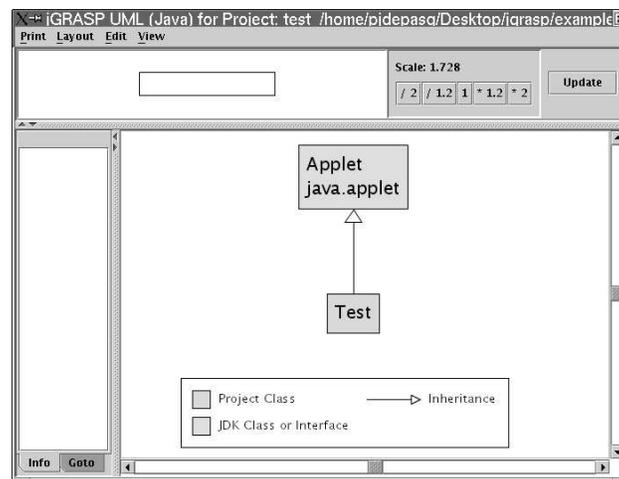


Figure 2.12: The jGrasp development/visualization environment displaying a corresponding UML diagram.

Currently, there is no reported data regarding the usage of the jGrasp programming environment by student programmers.

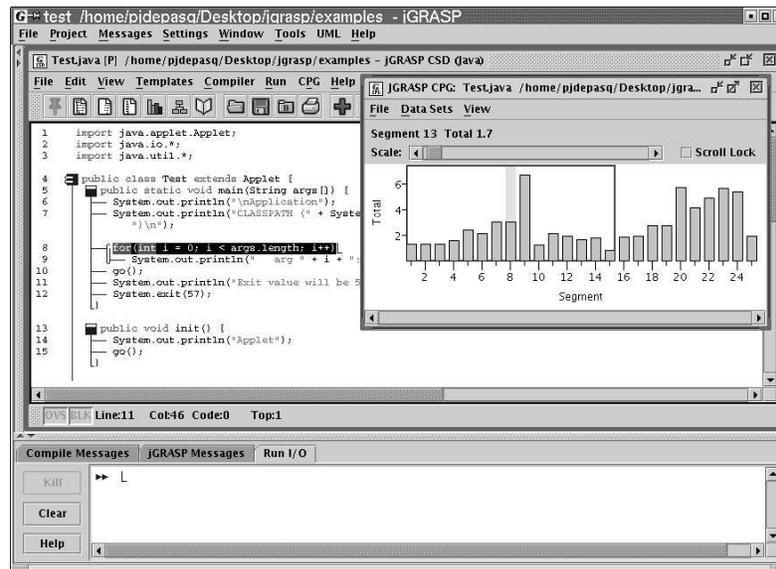


Figure 2.13: The jGrasp development/visualization environment displaying a corresponding Complexity Profile Graph.

2.3.4 BlueJ

BlueJ [40, 41] is an program development environment for the Java programming language which contains an editor, compiler, and a debugger. What makes BlueJ unique is the graphical representation of executing objects with which a user can interact. That is, the user can instantiate objects (in the runtime environment), see them represented graphically and then send messages interactively to them (call methods contained within the object) or have objects interact with each other. Figure 2.14 shows a screen capture of the primary development window of the BlueJ environment. The upper display area shows a Unified Modeling Language [57] (UML)-like class-view of the system under development. In the figure, the use of one or more Canvas objects by the Square, Circle and Triangle classes is denoted with a dashed arrow indicator. Inheritance can also be shown in this region simultaneously (none is displayed in this screen capture).

A user-instantiated Square object named *square_1* is shown in the lower display region of the main window. The user can interact with this object (see Figure 2.15) in real-time, calling its methods or inspecting the values of its fields. These interactive options are selected by accessing menu choices from the instantiated object in the lower display area.

In addition to its unique approach of an environment in which a user can instantiate and interact with objects, the BlueJ environment touts a simplified graphical interface. The developers claim only a 20 minute tutorial is needed by typical students to be up and running with BlueJ.

BlueJ attempts (with a good deal of success) to change the level of abstraction that users face while using a program development environment to produce Java programs. As with many programming environments, the users must deal directly with the file level while editing source code. BlueJ seeks to modify the file view to one of an object-based view, where the users can codify, compile, and manipulate objects directly (interactively).

Moreover, by having the BlueJ programming environment interact directly with the Java runtime

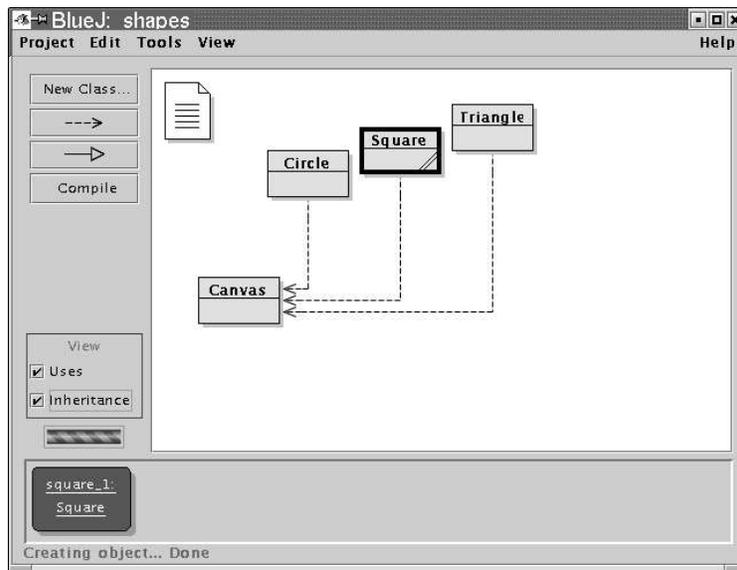


Figure 2.14: The BlueJ Java program development/visualization environment

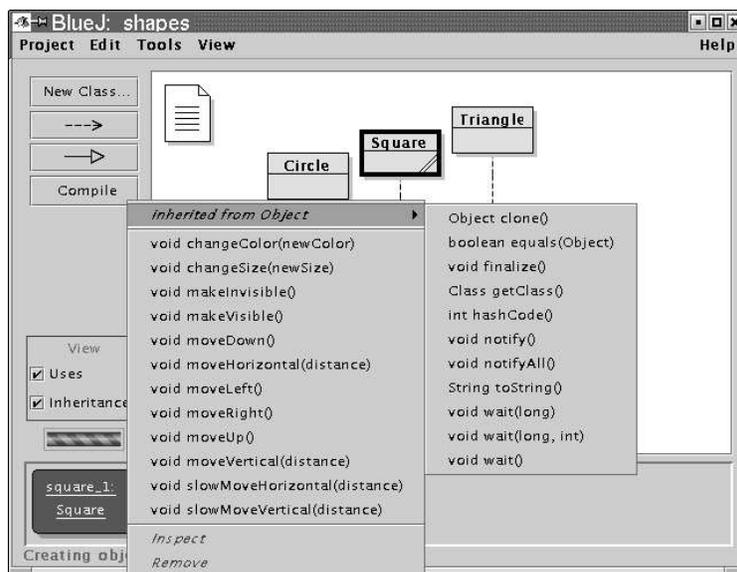


Figure 2.15: Interacting with an instantiated object in BlueJ

environment, some tricky linguistic issues can be avoided. Specifically, a main method in Java *must* be in the following form:

```
public class SomeClass {
    public static void main (String[] args) {
    }
}
```

Providing instruction about this code at the beginning of a course (so that the student understands where their program begins) may involve a discussion on visibility classes, modifiers, static methods, arrays and void return types. Creation and instantiation of Java objects directly in the BlueJ programming environment (with the help of the runtime system) avoids the need to implement a main method and increases the level of abstraction to the user. Thus, the programming environment itself supports the abstraction of the initial tricky concepts (and discussions) and encourages the discussion to focus on the objects-first approach.

One of the criticisms of the BlueJ environment is that it focuses less on program development and more toward the visualization of the object-oriented paradigm. Additionally, the environment abstracts the object-oriented paradigm to such a high degree that one could argue that the programming environment abstracts away the concept of a single method of execution of the resulting program. Of the eight demonstration projects packaged with the BlueJ installation (version 1.3.0 beta), only one contained a static main method for starting a Java application in the conventional sense.

One of BlueJ's authors has developed a corresponding textbook [7] which utilizes examples in BlueJ throughout the breadth of the text. The approach of the textbook is to present the material in the objects-first approach to teaching introductory programming. In doing so, examples in the text follow the objects-first approach by demonstrating solutions by directly interacting with the objects with BlueJ (as opposed to developing solutions whose interaction starts with execution of the `main` method).

More than half of the book is dedicated to this style (code development and execution-through-object-interaction) before execution outside of the BlueJ environment is demonstrated. However, the developers and proponents of the BlueJ system argue that this delayed approach to the introduction of the `main` method is not detrimental to the development of the student's comprehension of the creation of Java applications.

Currently, there is no reported data regarding the usage of the BlueJ programming environment by student programmers.

2.3.5 Ready to Program

Ready to Program⁶ [38, 39] is a simplified programming environment for Java development. Created by IBM and Holt Software, the main purpose was to reduce the complexity of the interface for an integrated development environment.

Ready to Program stresses the removal of the project management paradigm found in many modern IDEs. This is accomplished by having the user work directly with source code files that are not part of a larger project schema, and can be compiled directly. When the learner is ready to execute their program, they only need to hit the *run* button while viewing the source code file that contains the required main method for starting a Java program.

⁶Ready to Program is a registered trademark of Holt Software Associates, Inc.

Ready to Program provides a basic text editor with syntax color highlighting (see Figure 2.16) and only a minimal set of text-based buttons which allow the student to compile, load or save a source code file or execute a Java program. It is currently released in conjunction with J.N. Patterson Hume and Christine Stephenson's *Introduction To Programming In Java* textbook published by Holt Software Associates [39] in the fall of 2000.

```

CodingFrame[1].java - Ready to Program
File Edit Search Mark Run Help
Run Pause Open Save Indent Print Cut Copy Paste Find Find Next Replace
package sandbox.client;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import sandbox.compiler.*;

/**
 * $Id: CodingFrame.java,v 1.17 2001/03/23 16:04:02 ryturmer Exp $
 */
public class CodingFrame extends JFrame {
    private JEditorPane codePane=null;
    private JList errorList=null;
    private DefaultListModel errorListModel;
    private JToolBar commandToolBar=null;
    private JSplitPane splitPane=null;
    private JButton goButton=null, stopButton=null, openButton=null;
    private JButton saveButton=null, levelsButton=null, aboutButton=null;
    private JScrollPane codeScrollPane=null, errorScrollPane=null;
    private static String frameTitle = "CS1 Sandbox - ";
    private CodePaneListener codepaneListener = new CodePaneListener(this);
    private CodePaneCaretListener caretListener = new CodePaneCaretListener(this);
}
CodingFrame[1].java: 258 Lines
Line 1 of 258 Col 1

```

Figure 2.16: The Ready To Program development environment for Java

Ready to Program installs an IBM Java compiler and runtime environment, so there is no dependency on another installed development platform. Ready To Program achieves the intended design of a simplified interface in a educational-targeted programming environment. The reduced number of menu items and toolbar buttons is highly noticeable over commercial professional-strength Java development environments.

Ready to Program was only released in the fall of 2000, thus there has been little discussion of the results of using the environment. However, in a LISTSERV mailing to the Association of Computing Machinery's (ACM) Special Interest Group on Computer Science Education (SIGCSE), Byron Weber Becker noted the following response in soliciting opinions of Java programming environments:

"It, like BlueJ, is developed specifically for teaching and students. Ready is a more traditional environment than BlueJ. It's about as simple as interface as you can get."

Currently, there is no reported data regarding the usage of the Ready to Program programming environment by student programmers.

2.3.6 Alice

Alice [17, 16] is a three dimensional virtual world use to teach introductory 3D graphics authoring to novices. Developed at Carnegie Mellon University, the Alice environment allows students to create, and script virtual worlds containing a series of three dimensional objects, such as people, animals, tables, chairs, etc. Alice has been under development since 1995, and despite having undergone several revisions, its central theme has remained consistent - "describing the time-based and interactive behavior of 3D objects".

Students can utilize the Alice interface (see Figure 2.17) to place objects into the virtual world. Then, using a drag-and drop interface, users can program a series of serial or parallel actions which the objects will perform by dragging object method calls to a programming editor. In this fashion, the users are programming the events in the virtual environment. At any point, the users can play the sequence of the time-based events, viewing their programmatic results in the animated environment (see Figure 2.18).

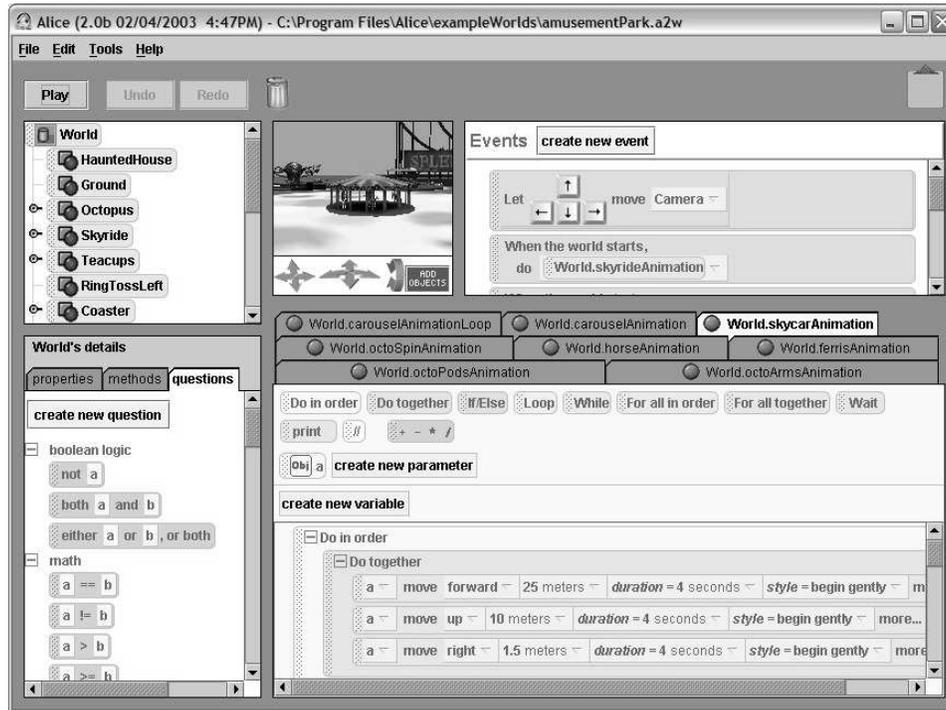


Figure 2.17: The Alice programming environment

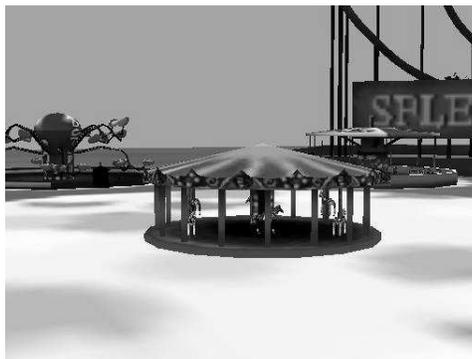


Figure 2.18: A view of a sample 3-D “world” created by the Alice programming environment. Here, the amusement park objects are in motion (spinning, turning, etc.) in parallel.

Alice was previously billed as an environment for graphics programming or virtual world creation. However, with the 2003 release of Alice, the focus has moved to using the environment as a vehicle for teaching introductory material about programming. Alice is focused around the objects contained within the virtual world and the methods and states that each object provides. Users can inspect each object, change its properties (direction, speed of motion, etc.) and modify the execution of their program. Alice strives to use the animation of program execution as the vehicle for the students to assimilate how the objects work individually and potentially together to complete a given task.

Events in Alice are consistently time-based, each taking one second to complete. To accomplish parallel events (animation), incremental motions are calculated for each object over a given interval and then interleaved in a round-robin fashion. By doing so, Alice can support parallel task programming (thereby enhancing the "real-world" effect) without introducing programming syntax.

Alice uses a propriety language to reference its objects and methods. In addition, the implemented language that the users learn supports functions, decisions, recursion, looping, and events/interactions. To simplify coordinate system references, each object has their own relative coordinate system and common language words (references) are used to support commands.

For example, a rabbit object may be told to move forward by two meters by using creating a program statement such as:

```
bunny.Move(forward, 2)
```

Alice comes with a series of tutorials to help users learn the system. The interface can be rather complex and cumbersome, however, the tutorials introduce the system and the user interface's functionality in a structured methodology. Currently, there is no reported data regarding the usage of the Alice programming environment by student programmers.

2.3.7 CodeLab

CodeLab [80] is a web-based programming language practice environment for learning Java, JavaScript, C, and C++ produced by TuringsCraft, Inc. Similar to course textbooks, the CodeLab environment breaks coverage of the language into distinct chapters, each with their own exercises to solve. Exercises tend to be focused on individual concepts or language features which directly fall within the chapter topic, with each language chapter containing fifty to one hundred and fifty exercises.

Students using the CodeLab can select with which chapter to work, and then also select which exercise to attempt. Once selected (see Figure 2.19) the student is presented with a problem statement and a solution area. Problem statements progress from requiring the implementation of one language statement (e.g. defining a specified identifier), to several statements (implementing a loop), to authoring entire functions/methods.

Upon completion of an exercise, the student must submit their answer to the server for grading (see Figure 2.20) where the result is displayed for the the student. If the solution is judged to be incorrect, the student can either view a correct solution or attempt the exercise again.

Feedback to the user varies based on the result of the student's attempt. That is, in the event of a logic or runtime error, the resulting output from the programming environment may be brief (see Figure 2.20). However, in the case of compilation errors, the resulting output is more detailed, including a discussion of the compiler's error message (see Figure 2.21). Note that in this case the wrapper code (used to wrap the student's submission for compilation) is also displayed, permitting the student to view the entire context of the compilation error.

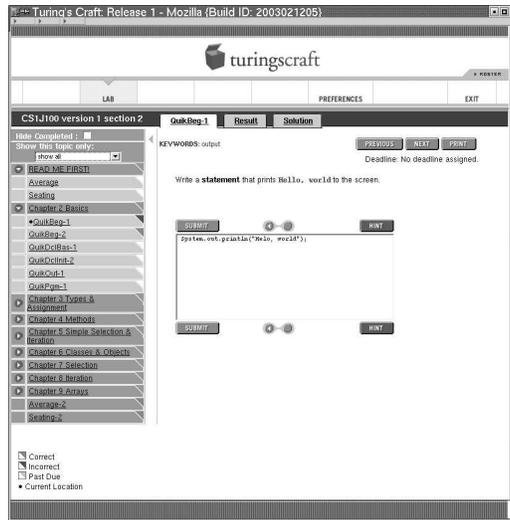


Figure 2.19: Attempting a problem in the CodeLab programming environment



Figure 2.20: Obtaining an exercise result from the CodeLab programming environment



Figure 2.21: Obtaining an error message from the CodeLab programming environment

CodeLab is designed to be inserted into the course curriculum for introductory programming classes. The environment provides support for maintaining class rolls and data collection related to each student's performance (submissions and results). Instructors also have the ability to assign due dates to individual exercises, thus utilizing the CodeLab exercises as homework assignments. Currently, there is no reported data regarding the usage of the CodeLab programming environment by student programmers.

2.3.8 ProgramLive

ProgramLive is a multimedia textbook authored by David and Paul Gries [34] for learning Java and object-oriented programming. The ProgramLive software graphically reproduces a book paradigm for the students (complete with chapters, sections, page tabs and a spiral bound edge). ProgramLive contains narrative presentations using recorded audio tracks to present programming concepts to the learner.

Other features of the Gries effort include in-text quizzes, a glossary linked to the chapter prose by hypertext links, an index, and the ability for the learner to set a bookmark and return to it at a later point. One of the most interesting features is the *code tool* section, which is a simulated program development environment. The code tool features a panel of Java source code which is animated to a voice narration and a textual and graphical presentation in a second window. Users can enter this section to observe code development complete with a narrative track which can be stopped and repeated.

A more complete analysis of the ProgramLive system was not possible utilizing the demonstration disk made available for review. Attempts failed to obtain a complimentary copy for examination purposes. It is hoped that complete review can be undertaken in the near future as the ProgramLive environment seems to have promising aspects to it. Currently, there is no reported data regarding the usage of the ProgramLive programming environment by student programmers.

2.3.9 MiniJava

In 2001, Eric Roberts of Stanford University introduced MiniJava [63] at the ACM's annual symposium on computer science education. MiniJava is subset of the Java programming language designed to reduce the complexity of the language – especially for use by novice programming students.

In creating MiniJava, Roberts starts with a release of the Java development environment and reduces the base language by modifying it as follows.

- Reduces the size and content of the class library – MiniJava reduces the number of supporting library classes from 705⁷ to just 17 classes. Additional classes can be added to extend the core library MiniJava provides.
- Eliminates support for inner classes – Roberts argues that inner classes in Java “...add no fundamental power to the language and represent a significant complication for new students.” [63].
- Eliminates several control statements – MiniJava removes several control statements, including:
 - **do-while** loops (“More often than not, the resulting programs are buggy since they fail in those cases in which the loop should not be executed at all.” [63]),
 - **continue** statements (“...adds nothing to the language and simply encourages poor programming style.” [63], and
 - fall-through behavior in **switch statements** (treated as a run-time error by MiniJava rather than an obscure language flaw)

Following the creation of the base subset of Java that MiniJava provides, MiniJava is then supplemented with several extensions specifically designed to support novice learners. The key extensions added to MiniJava are:

- implementation of a read-eval-print-loop (REPL) – as with other recent environments (see Sections 2.3.1, 2.3.4, and 2.3.2), MiniJava has implemented a REPL workspace used to issue Java expressions and statements. Students can instantiate objects and call methods from within the workspace (known as the console in MiniJava).
- console support – MiniJava provides a Console class to ease input and output to the REPL console (mentioned above). This is particularly useful for teaching input handling to novice students as Java's implementation is non-trivial and requires the use of several classes (including stream and buffering classes). MiniJava supports only the reading of integers, doubles, and lines (strings) in the simplified Console class.
- base application classes – Two classes (Program and GraphicsProgram) are provided to support students creating fully functioning programs, once they have moved away from using the REPL console (see above). These classes provide the framework for creating text-based (Program) and graphical-based (GraphicsProgram) programs by creating the input/output console and graphics framework (respectively) for the students.

In the case of the Program class, a simplified **main** method is called automatically at run-time, allowing the students to avoid the implementation of the standard **main** method in Java – **public static void main(String[] args)**. The students now implement their program's starting point as:

⁷Roberts reports that one branch of the library package tree contains 705 public classes usable by Java programmers. Updated releases of the Java programming language have increased this count even higher.

`void main()`. This delays instructional coverage (or visibility of such items) of Java's visibility modifiers (`public`), class/instance modifiers (`static`), and parameter passing or arrays (`String[] args`).

- primitive types are objects – Unlike Java's types, primitive types in MiniJava are objects. In MiniJava, the primitive types are placed in their corresponding Java wrapper classes, helping to avoid confusion by the students of the difference between a Java primitive type and its wrapper class.

Though introduced at the conference through his paper [63], a released version of MiniJava does not seem to be available at the present time. In his paper, Roberts indicates that MiniJava was still under development and was to be published in conjunction with a textbook (unnamed) in 2001. Efforts to reach Eric Roberts to obtain an update of the language/environment for this dissertation were not successful.

2.4 Summary

This chapter has concentrated on a review of the previous work where subsets pertain to programming languages as well as programming environments as related to this dissertation's research questions. It should be noted that not every prior programming environment ever created was examined due to the large number of prior environments, and the difficulty in obtaining versions compatible with today's operating systems.

Deek's [22] classification of twenty five systems (dating back to the mid 1970s) was particularly useful for encapsulating the goals and ideals examined in prior efforts. The paper categorizes each system into one of three categories: programming environments, intelligent tutoring systems, and intelligent programming environments. None of the environments presented by Deek included support for subsetting a programming language, though several are based upon mini-languages created specifically for their programming environment's implementation or a subset of a larger language (such as Pascal).

Table 2.1 summarizes the capabilities of the environments elaborated earlier in this chapter. Only four of these works (Cornell Program Synthesizer, Bittitalk, DrScheme, and MiniJava) include some type of support for subsetting the target language in order to better support novice programmers.

While each of the above listed languages and programming environments have strengths and weaknesses, none of the literature included quantitative analysis as the result of utilization by a novice programming population. The primary goal of this work seeks to fill this void with a quantitative analysis of the effect of the utilization of language subsets on novice student programmers at the collegiate level. This dissertation will establish what quantitative gains (if any) are realized by the implementation of instructor-controlled language subsets in a programming environment used by novice programmers.

Table 2.1: Summary of capabilities of the programming environments presented in this dissertation.

Programming Environment	Aids in visualization of executing code	Provides (live) object interactivity	Subset(s) of programming language present
Cornell Program Synthesizer (pg. 16)			Yes
Pascal Trainer (pg. 17)			
Emile (pg. 18)	Yes		
Solo (pg. 19)			
Bittitalk / ViewMatcher (pg. 23)	Yes	Yes, via Smalltalk programming environment	Yes
Karel / Karel++ / JKarelRobot (pg. 24)	Yes	Some implementations do	
DrScheme (pg. 26)		Yes	Yes
DrJava (pg. 28)		Yes	
jGrasp (pg. 30)	Yes		
BlueJ (pg. 32)	Yes	Yes	
Ready To Program (pg. 34)			
Alice (pg. 35)	Yes	Yes	
CodeLab (pg. 37)			
ProgramLive (pg. 39)	Yes		
MiniJava (pg. 40)		Yes	Yes

Chapter 3

The *CS1 Sandbox* and Supporting Software

3.1 Overview

To support the experiment this dissertation encompasses (see Chapter 4), a Java-based [33, 75] prototype C-programming environment was developed and deployed. Design and implementation work began in the Fall of 2000 and was completed during the summer of 2002 - in time to support the operation of the experiment during the Fall semester of 2002.

During the development time, the implementation team was comprised of this author as project manager and a cadre of undergraduate and graduate students from the Computer Science department. These students enrolled in independent study courses and provided software design, development, and testing.

In nearly every case, each student was assigned a series of particular tasks to accomplish, many in the form of software components to build, each adding functionality to the system. As shown in Table 3.1¹, the number of students working during a particular semester varied. This variability was due to the success of the recruiting efforts by the project manager to solicit such students, and the general need for student assistance at the time.

Table 3.1: Student Developer Population By Semester

Semester	# of Students
Fall 2000	2
Spring 2001	6
Summer 2001	2
Fall 2001	7
Spring 2002	2
Summer 2002	1
Fall 2002	2

The applications that were developed resulted in more than one hundred and thirty source code files, and over thirty seven thousand lines of Java source code. The entire project was managed using the Concurrent

¹Excludes the project manager who developed software throughout the life of the project.

Versions System (CVS) source code control system [30], allowing the users to concurrently develop individually and archive their source code modifications simultaneously through a central system.

The model of the *CS1 Sandbox* applications is that of a client-server paradigm. Subjects used the *CS1 Sandbox*'s programming environment as a stand-alone client-side application. The environment is a client in the sense that it may (in the versions that support doing so) download and apply subset configuration files from a subset configuration file server. The environment also sends data collection information to the same processor which acts as a data collection server.

In the sections that follow, the client portion of the *CS1 Sandbox* is first discussed (see Section 3.2). Section 3.3 covers the details of the subset configuration file server application. Finally, Section 3.4 details both the client and server-sides of achieving the experiment's data collection process.

3.2 The *CS1 Sandbox* Programming Environment (client)

What is known as the client-side of the *CS1 Sandbox* is truly the programming environment prototype developed for this dissertation. The programming environment has been divided in to several parts (each detailed in their own section of this chapter) according to their functionality.

3.2.1 AAnother Tool for Language Recognition (ANTLR)

The heart of the *CS1 Sandbox* programming environment (the lexer, parser, semantic validator, and the interpreter) is built upon the ANTLR [60] translator generator – a freely available (see <http://www.antlr.org>) Java-based generator which can create Java, C++, and C# output for inclusion in a translation system. ANTLR uses LL(k)-style analysis in the generation of translator(s) and is designed to be incorporated easily within a larger application framework when a recognizer is required.

3.2.1.1 ANTLR Basics

ANTLR was selected as the language recognition-generator for the *CS1 Sandbox*'s underlying lexer, parser, semantic validator, and interpreter for the *CS1 Sandbox* programming environment. Despite considering other systems, such as Lex (Lexical Analyzer Generator) and YACC (Yet Another Compiler-Compiler), ANTLR was selected for two principal reasons. Firstly, ANTLR generates Java source code for inclusion into a larger project. This was a highly desirable feature for any generation system we would select, as the programming environment was to be developed and deployed using Java (so that ultimately it would be available on a wider variety of platforms).

Secondly, ANTLR provided something the other systems did not – the framework of the interpreter. ANTLR's ability to generate Abstract Syntax Tree (e)-walking software routines permitted the development team to easily modify the generated software creating both a semantic validator and an interpreter more easily than compared to using Lex or YACC (or other solutions).

To create each component of the interpreter's system (a lexer, a parser, a semantic checker, and an interpreter), a single input file was provided to the ANTLR software during the build of the *CS1 Sandbox*. Each input file was processed by ANTLR and resulted in the production of a single Java source code file which was then included in the *CS1 Sandbox* software. For example, a file named `CLexer.g` was created as input to ANTLR for creating the *CS1 Sandbox*'s lexer. ANTLR processes the input file and produces a corresponding Java source code output file (`CLexer.java`) for inclusion with the build of the *CS1 Sandbox*

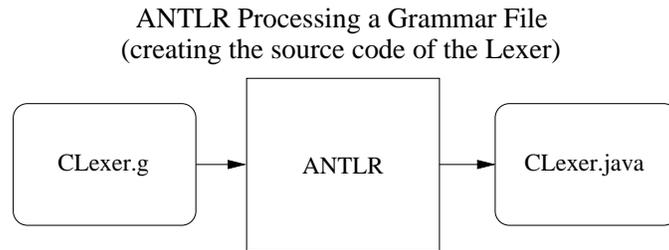


Figure 3.1: ANTLR role in processing a grammar file and producing compilable Java source code for inclusion in a larger project.

application (see Figure 3.1). This process is repeated for each component of the interpreter’s system – `CParser.g`, `Semantic.g`, and `Interpreter.g` were all processed by ANTLR producing the `CParser.java`, `Semantic.java`, and `Interpreter.java` files.

ANTLR’s grammar files vary depending on the type of component one wishes to build. For instance, the lexer input file `CLexer.g` defines both the tokens which should be located on the lexer’s input stream (a subject’s source code to a program), as well as embedded Java programming statements which are passed through to the corresponding output (the lexer’s source code file). The bulk of this file for the *CS1 Sandbox* project contained token definitions (refer to Appendix J.1).

The grammar file for the parser is slightly different, as it contains a pattern-matching grammar used to identify sequences of tokens coming from the lexer. Once a pattern is identified (C++ language statements) an entry into the AST s. Similar to the lexer’s grammar input file to ANTLR, the parser’s input file contains embedded Java statements (initialization routines, debugging statements, etc.) which are also included in the Java output file that ANTLR produces.

The grammar files for both the semantic validator and interpreter subsystems are slightly different ever still. Given that the output of the parser is an AST tree, the main functions of the semantic validator and interpreter are to walk the AST tree (looking for errors in the case of the semantic validator, and executing the program in the case of the interpreter). To accomplish this, ANTLR defined a grammar to perform the tree-walking task. These grammar files were also a pattern-matching grammar, as was the parser. However, rather than matching lexer tokens, the tree-walking grammar matches AST structures (nodes in the AST) and performs corresponding actions upon matches. Java statements are embedded within the grammar which define the actions performed when a structure match occurs. As with the other components derived from ANTLR (the lexer and the parser), embedded Java statements are carried forward by ANTLR into the output ANTLR produces. Each grammar “rule” in tree-walkers results in a corresponding Java method (function) which contains the Java statements embedded within the grammar input file.

In this fashion the developers were able to specify the grammar of the language to be accepted by the programming environment (see Appendix I for a full listing of the implemented grammar), and ANTLR would then build a specific component to suit the needs of the task at hand.

3.2.1.2 ANTLR’s Integration into the *CS1 Sandbox*

Four distinct ANTLR input files (also known as ANTLR grammar files) were developed by the programming team and used to generate (through ANTLR) a lexer, parser, semantic validator, and interpreter exclusively for the *CS1 Sandbox*. Each component is discussed in detail below and includes a

sample from their grammar file (input to ANTLR). A pictorial representation of the client-side of the *CS1 Sandbox* programming environment is shown in Figure 3.2.

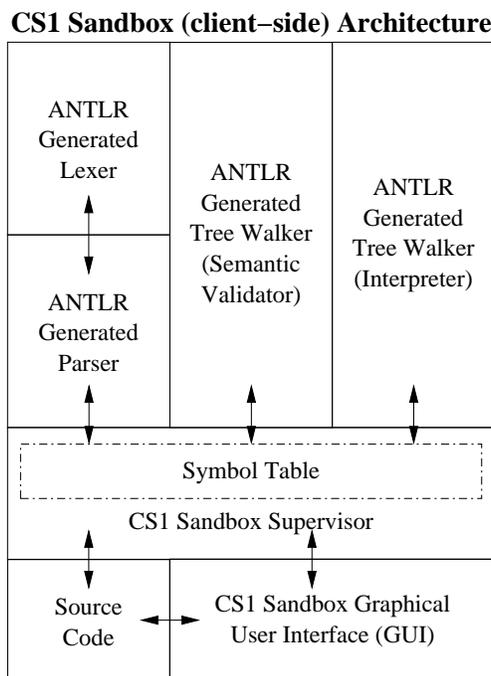


Figure 3.2: An architectural view of the *CS1 Sandbox* programming environment (client-side). The arrows indicate communications between the various subsystems of the application.

3.2.1.2.1 Lexer/Parser As with any scanner, the *CS1 Sandbox* lexer recognizes the vocabulary of the target language. This is achieved by breaking the input stream (in the case of the *CS1 Sandbox* the source code input) into a series of tokens for use by the parser.

Of particular note in the lexer are several issues. First, keywords in the grammar are tagged for special handling in the lexer. However, primitive data type names (short, int, double, etc.) are not handled in this fashion. These data type names are placed in the interpreter's symbol table directly by the software that handles the type management system (performed during the semantic check and interpretation phases). Therefore, initially data types appear to be identifiers when parsed, but are confirmed as keywords when they are detected in the symbol table.

Another interesting point about the lexer is the handling of C/C++ pre-processing directives. Students in CS1044 must rely on C/C++ library files to complete their laboratory programs and projects. Ordinarily, a compiler would utilize on a separate program to parse the pre-processing directives and respond or replace them accordingly. The *CS1 Sandbox* needed to support the limited set of directives the subjects would be utilizing during the experiment, and thus they were handled in the lexer (the loading of the header files' symbols) and the parser (validation of the current subset permitting the use of the specified

header file). (Table 3.2 defines the pre-processing directives that the *CS1 Sandbox* programming environment supported.)

Table 3.2: Pre-Processing Directives Supported by the *CS1 Sandbox*

Name	Description
climits.h	contains constant identifiers of integer data type limits (e.g. INT_MIN, INT_MAX)
cstdlib.h	contains standard C library functions (e.g. atoi, labs, etc.)
fstream.h	contains input and output file stream representations and support functions
iostream.h	contains the cin and cout stream representations and support functions
iomanip.h	contains output formatting stream manipulators
math.h	contains math-related functions (cos, sin, tan, etc.)
string.h	contains string manipulation functions (length, concat, etc.)

The parser built for this project (through the use of the ANTLR toolkit) obtains tokens from the lexer and processes them with the following two goals in mind. First, the overall job of the parser is to build an abstract syntax tree as the output of the parser. The second goal is to enforce the language subsets imposed upon the client programming environment (if applicable).

The parser operates by successively calling the lexer to obtain tokens, and validates a stream of tokens as a valid statement in the grammar that the *CS1 Sandbox* supports (see Appendix I). Only valid syntactically correct statements are added to the e, while those incorrect statements generate an error message in the *CS1 Sandbox*'s error list.

During the validation phase of processing a statement, the parser cross-checks the statement with the *CS1 Sandbox*'s restriction management software to ensure the statement may be used by the programmer. If the statement in question violates a subset restriction, an error message is created and added to the error list (in an identical fashion to those normally generated by parser when validating the syntactic nature of a statement).

Appendix I provides a full listing of the language grammar(s) supported by the *CS1 Sandbox* programming environment. Appendix J.1 lists the ANTLR-grammar file used as input to ANTLR to produce the lexer software, while Appendix J.2 provides the corresponding ANTLR output (Java source code) file. Appendices J.3 and J.4 provide listings of the ANTLR input and output files (respectively) used to create the *CS1 Sandbox*'s parser.

3.2.1.2.2 Semantic Checker/Interpreter Construction of the semantic checker and the interpreter were not unlike the lexer and parser. Statement-specific functionality was embedded into the semantic checker's input which was then passed as part of the output from ANTLR via processing the input file. In this fashion, the developers were able to embed calls to classes, objects, and methods outside of the ANTLR toolkit, which existed elsewhere in the build of the *CS1 Sandbox*. A small sample of the semantic checker's grammar file (input to ANTLR) showing the validation of the *if* statement is shown in Figure 3.3.

In Figure 3.3, an `ifStatement` node in the AST is verified to ensure that it is followed by no less than two child nodes—an expression node and one or two statement nodes. The expression node is the boolean conditional expression to be evaluated during execution, while the first statement node represents an AST subtree of the true clause of the *if* statement. The second statement node represents the optional `else` clause of the *if* statement.

Each child node in the AST may be of a form which in itself is another node in the tree, as is the case here with the expression node and statement node(s). Once these child nodes are identified, the semantic

```

//
// Semantic validation of 'if' statement
//
ifStatement
{
  // Create a temporary object and set it to null
  AbstractType type=null;
}
// The AST here should be of the form:
// + IF node
// | |
// | + EXPRESSION node
// | |
// | + STATEMENT node
// | |
// | + STATEMENT node (optional)
// |
// + next AST node
//
: #(ifNode:IF type=expression statement)? {
  // Test the node named 'type' and ensure it is a valid
  // test expression (returns a boolean value)
  if (!validTestExpression(type)) {
    // If this is not a valid text expression, report this
    // as an invalid boolean expression.
    reportError("", ErrorCodeTable.INVALID_BOOLEAN_EXPRESSION,
      ((SandboxAST)ifNode).getLine(), 0);
  }
}
;

```

Figure 3.3: An example of a semantic validation of the `if` programming statement as input to ANTLR. The example includes a validation check to ensure the statement is enabled for use (by the subset). Refer to Appendix J.6 for the ANTLR-generated Java source code.

checker will call their respective validation routines to ensure that they themselves are valid (e.g. ensuring that a valid expression exists).

Upon completion of the validation of the structure of the AST subtree rooted at the `if` statement, one or more lines of Java source code can be inserted to allow for additional validation commands. In the example in Figure 3.3, the `if` statement is further evaluated to ensure that the returned type of the embedded expression is a valid test expression (one that results in a boolean result). In this case, if the resulting expression type is not valid, an error message is generated with the call to the `reportError` method - where it is passed a null message, the specific error code, and the line number on which the error occurred.

As with the parser, the semantic checker ensures that restrictions placed upon the programmer are enforced. Here, the semantic checker verifies the status of any header files that the programmer has utilized, as well as each of the primitive data types supported by the programming environment. This validation involves the checking of a status flag in the restriction management subsystem for each header file or data type encountered. For each instance of a violation of the restrictions being found, a corresponding error message is generated for the user.

With semantic checking completed, the current error list is examined to determine if there were any outstanding errors the programmer needs to address. If errors are present, they are listed in the GUI's error list and no attempt at interpretation is attempted. If errors do not exist in the list, the interpreter is called and is passed the program's AST.

As with the semantic checker, creation of the interpreter subsystem was developed with the assistance of ANTLR, via the creation of an input file defining the structure of the grammar. Each defined statement contained execution directives to perform the execution of the program. Figure 3.4 shows a portion of the

ANTLR input for creating the interpreter subsystem – in this case, the interpretation directives for encountering the `if` statement.

```
//
// Interpretation of 'if' statement
//
ifStatement returns [java.lang.String exitCode]
{
    // Set the exitCode object to null
    exitCode = null;
}
// Alias this top-level node as 'ifNode'
: ifNode:IF
{
    // Print debugging string and create objects referring to
    // the sub-nodes from the 'ifNode'. The 'falseNode' is optional
    // and therefore may be null.
    debugStream.println("Executing if statement");
    AST exprNode = ifNode.getFirstChild();
    AST trueNode = exprNode.getNextSibling();
    AST falseNode = trueNode.getNextSibling();

    // Evaluate the expression and obtain the result
    boolean result = evaluateTestExpression(exprNode);

    // Follow the appropriate branch, depending on the result
    // of the expression.
    if (result) {
        debugStream.println(" Following the true branch:");
        exitCode = statement(trueNode);
    } else if (falseNode != null) {
        debugStream.println(" Following the false branch:");
        exitCode=statement(falseNode);
    } else {
        debugStream.println(" Following the false, but no branch exists.");
        exitCode = null;
    }
}
}
```

Figure 3.4: An example of interpretation directives of the `if` programming statement as input to ANTLR. Refer to Appendix J.8 for the ANTLR-generated Java source code.

Appendices J.5 and J.6 provide listings of the ANTLR input and output files (respectively) used to create the *CS1 Sandbox*'s semantic validator, while Appendices J.7 and J.8 list the input and output grammar files used to create the interpreter subsystem.

3.2.1.2.3 ANTLR/*CS1 Sandbox* execution sequence The *CS1 Sandbox*'s execution sequence (when attempting program validation/compilation and execution) is similar in nature to many other programming environments. Once the user has entered their source code into the graphical user interface, they can initiate a validation attempt. When the validation begins, the *CS1 Sandbox*'s supervisor subsystem instantiate a lexer and parser object (based on the code that was generated by ANTLR during the build process of the programming environment). The lexer is then passed the contents of the source code window and the parser begins obtaining tokens from the lexer and building an AST. These steps are reflected in Figure 3.5 denoted as step one. The output of the parser is a list of error messages which emanated from the lexer and parser (if applicable), and the AST of any successfully parsed statements from the submitted source code.

Next, the supervisor subsystem examines the error list from the lexer/parser. If the list is not empty, any error that was generated is prepared for display, passed to the GUI for display in the error list area (see

Section 3.2.4), and processing terminates (returning programmatic control to the GUI, await further user input/events). If the lexer/parser error list is empty, the supervisor subsystem proceeds by instantiating a semantic validator object, and passing to it for analysis the AST obtained from the parser. This phase is reflected in Figure 3.5 as step two.

As the semantic validator processes the AST, any warning or errors which are discovered are added to an error list (specific to the semantic validator). This error list is the only product of the semantic validator when it completes its analysis. Control is then returned to the *CS1 Sandbox*'s supervisor subsystem, which examines the error list obtained from the semantic validator. If the list is non-empty, it is prepared for display, passed to the GUI for display in the error list area, and processing terminates. If the error list is empty, then the AST is judged to be “correct”, an interpreter object is created, and the AST is passed to the object for interpretation. Note that the AST does not change during the semantic validation phase, and the identical AST is used during the interpretation phase. This last phase is reflected in Figure 3.5 as step three.

Similar to the previous phases, if the interpreter subsystem discovers run-time errors while processing the AST, error messages are created and inserting into the error list and processing continues. When the interpreter completes, an error list is returned from the interpreter to the supervisor subsystem. Any messages placed in the error list during interpretation will be prepared for display, passed to the GUI for display in the error list area, and control will return to the GUI – awaiting additional user input/events.

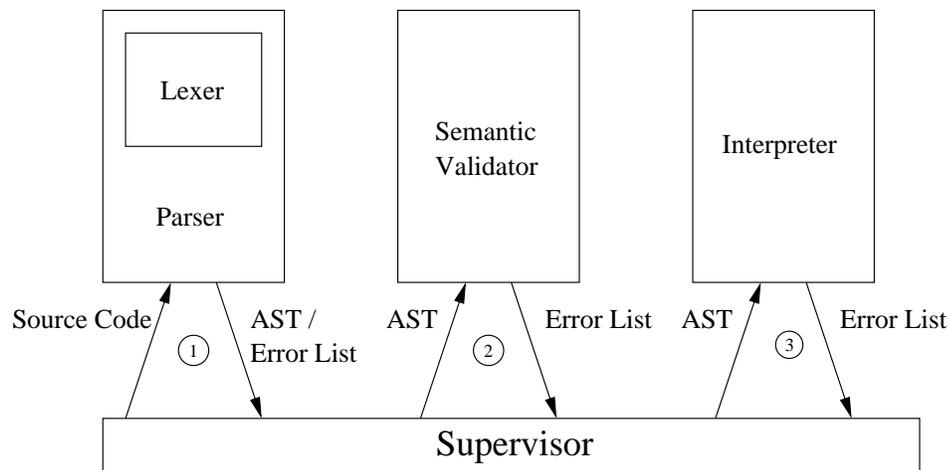


Figure 3.5: The calling sequence of ANTLR within the *CS1 Sandbox*

3.2.2 Input and Output

Due to the heavy reliance of both input and output streams in the CS1044 course at Virginia Tech (where the *CS1 Sandbox* programming environment would be utilized), the *CS1 Sandbox* would therefore need to support C++-style input and output file streams. For all closed-laboratory and out-of-laboratory programming projects, students are required to pass all input to their programs via a file stream (using files). Similarly, file output was the required mode of operation for all graded work. This reliance upon the C++-style streams comes from the prevalent use of the Curator system used to grade the work of each student (see Section 4.2.2). In addition, input and output to the C++ standard input and output streams

(`cin` and `cout`) were also supported through the use of the *CS1 Sandbox*'s input/output window (see Figure 3.6).

3.2.2.1 Input

Input at run-time to the *CS1 Sandbox* is achieved via two methods: keyboard interactive (through the input/output window) or file-based (utilizing C++-style file streams). The focus of the input work was on the file-based approach due to the heavy attention the file streams would receive by the subjects when completing their laboratories and projects. However, for pedagogical reasons the development team felt it was important to implement the interactive-based approach as well as the stream-based approach.

Both approaches used a buffer to read the input from the origin. This was particularly important to handle the stream manipulations required by supporting the input (extraction) and output (direction) operators (`>>` and `<<`, respectively). However, another reason led us to implement the streams via buffers. The development team anticipated needing to support the language commands to place data back on the streams (the putback stream method), as it was referenced in the course literature. To successfully implement such a specification, we anticipated relying on the buffer to help accomplish this. However, this feature was never finished before the programming environment was used for the experiment, and the closed-laboratory and out-of-laboratory projects were careful to avoid requiring the need to push data back onto an input stream.

In the case of interactive input, the incoming text was first echoed to the output display window when the user pressed the Enter key on the keyboard and was then placed in the buffer for later processing. This behavior mimicked that of interactive input in Microsoft's Visual C++ version 6.0

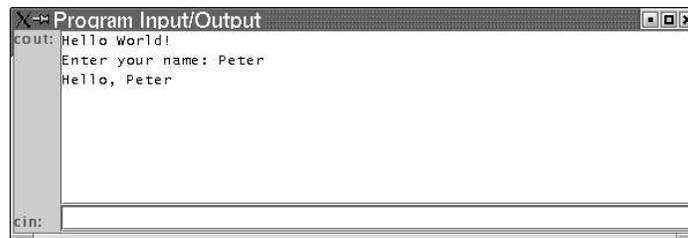


Figure 3.6: The *CS1 Sandbox* Programming Environment's input/output interface

3.2.2.2 Output

The *CS1 Sandbox* supports both C++-style file output streams as well as the standard output file stream (`cout`). File output streams were able to open a new file for writing given one key constraint – the default file location was the directory in the filesystem where the executing program's source code resided. Clearly, this required the programmer to be able to write to this filesystem location. In order to promote additional filesystem security, the *CS1 Sandbox* file stream open function calls did not support the naming of a file which utilized a directory path in the filename. That is a user was forced to open and access only files in the same directory in which the source code originated.

As with the input support, the *CS1 Sandbox* supported interactive output (to the input/output interface) by echoing any output bound for the standard output stream to the `cout` portion of the input/output interface.

3.2.3 Graphical User Interface

The user interface of the *CS1 Sandbox* is shown in Figure 3.7. From its inception, it was designed to be uncluttered with respect to the number of menus, buttons, and text areas (as compared to Microsoft’s Visual C++ Version 6.0/.NET).

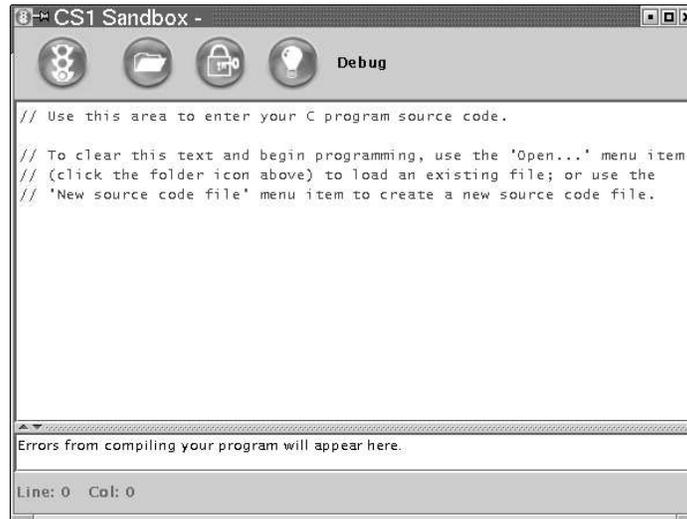


Figure 3.7: The *CS1 Sandbox* Programming Environment

The student’s interface features one button and three menus, as well as one text editor region for entering source code, a scrollable list for errors, and a textual display for the presentation of cursor location information. The interface shown in Figure 3.7 is identical to that used by the subjects throughout the experiment, with one exception. Figure 3.7 shows an additional “Debug” menu which was included for the project development team. In the version of *CS1 Sandbox* that was provided to the subjects, this particular menu was hidden from view.

The buttons and menus provided (from left to right above the source code entry area) are as follows:

- “Go” button - This button is shown with the icon of a traffic signal with the green lamp lit. This button is used to initiate the parsing of a program contained in the text area just beneath the row of buttons and menus on the interface. When the translator is active, the button’s icon is changed to reflect a traffic “Stop” sign, indicating that the button can be pressed again and the translator terminated. If the translator successfully completes the parsing of the program, and no errors are discovered, interpretation of the program immediately commences without a change of the button’s image (at this point it should still be a “Stop” sign). Upon completion of the interpretation (or in the case of termination of the translator due to errors), the “Stop” sign icon is replaced once again with the “Go” traffic signal icon.
- “Folder” button - The “Folder” button is displayed to the right of the “Go” button and is shown with the icon of an open folder. This button (when clicked) leads to a menu of file options, (New..., Open..., Save, and Save As...) as well as an Exit menu item to terminate the programming environment.

- “Subset” button - This button is shown with a lock and key icon denoting the availability to unlock additional features. For subjects who were required to use the *CS1 Sandbox* without subsets, this button was disabled (grayed out) and inaccessible. Subjects required to use the *CS1 Sandbox* with subsets had this button enabled by default. When depressed, the “Subset” button displayed a menu permitting users to access the portion of the interface which permitted the downloading and application of a language subset, as well as a window which was used to display the current language subset settings (see Section 3.3.2).
- “Lightbulb” button - The final button (shown with a lightbulb icon) is used to access the “About this software...” menu item. Originally, additional entries were to be added to this menu (creating a help menu), however these efforts were scaled back for other development tasks. When selected, the “About this software...” menu item displays a small window use to provide information about the current version of the *CS1 Sandbox*, the list of developers, and links to software utilized in creating the programming environment. These items were displayed in the window using a tabbed pane (see Figure 3.8).
- “Debug” menu - The “Debug” menu provided access² to the *CS1 Sandbox* developer’s debug window. This interface contained three text areas for the output of debugging message from the following sources: the semantic validator, the interpreter, and the Java Run Time Environment’s standard error stream (see Section 3.5.1).

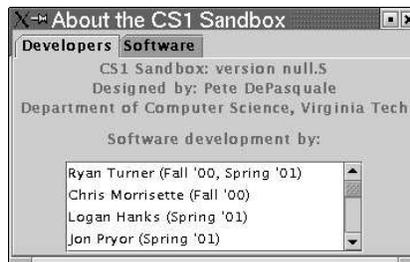


Figure 3.8: The *CS1 Sandbox* Programming Environment’s About window

Below the area used to contain the buttons/menus on the interface, is the main text editing area. The text area is a simple region used to enter the source code of a program by a user. The area itself is relatively unsophisticated in that it does not support a search and replace option, syntax color highlighting, or other features generally found in programming environments.

To maintain the simple design of the interface of the *CS1 Sandbox*, the development team decided to support the editing of only one source code file at a time. In doing so, this not only reduced the complexity of the interface to the user, but additionally reduced the complexity of interpreter’s systems as well (the team did not have to decide how to process multiple files, or provide an interface to permit the user to make the decision of which file to process first).

Several modifications were added to the basic text area upon its integration with the *CS1 Sandbox*. These included the following:

- keyboard events - Support for keyboard-based events were added to the interface to permit a shorthand method for initiating an interpretation attempt. Additionally, keyboard-based access to debugging features used by the developers was added to the text area component.

²The “Debug” menu is only available via the developer’s release of the programming environment.

- syntax color highlighting - As with most program development environments, there was a strong desire to incorporate syntax color highlighting capabilities into the programming environment. To accomplish this we integrated Stephen Ostermiller's Syntax Color Highlighting Java package [58] into the *CS1 Sandbox*. As delivered, the highlighter can parse a variety of formats including Java, C/C++, LaTeX, and others.

The development team configured the highlighter to recognize and highlight keywords from C++. The Ostermiller software was chosen primarily due to its unrestrictive license, because it was written in Java, would integrate into the environment without an exceeding amount of work, and because it handled the highlighting functionality for us.

- mouse events - Capturing mouse events in the source code editing text area was essential so that we could properly display the current line number and column number in which the mouse cursor is located.

Beneath the source code text area lies the error list region. By default, this area contains the text “*Errors from compiling your program will appear here.*” However, if after an interpretation attempt errors exist in the input source code, this region contains a listing of each error discovered. Each entry in the list is comprised of line and column numbers of the error (in the source code file), as well as a short message identifying the error. (See Figure 3.9 for a screen capture of the *CS1 Sandbox* interface with errors displayed.)

The user can select an error from the error list (using the mouse) affecting the interface of the programming environment in two ways. First, the line on which the error occurred is highlighted in the source code text area. If the offending line is not currently displayed, the text area is automatically scrolled into view. Secondly, a button appears in the lower right corner of the interface offering additional information to the programmer about the selected error (refer to Section 3.2.4 for more information about the help system in the *CS1 Sandbox*).

3.2.4 Help System

The help system embedded in the *CS1 Sandbox* is comprised of two components: the error messages (emanating from the lexer, parser, semantic validator, and interpreter) and the help display system (actual help files used to further explain an error situation to the user).

Generally speaking, error messages can be created by any of the underlying software (lexer, parser, semantic validator, and interpreter) used to process a user's program and then added to an error list. Following each phase of compilation/execution, the error list is displayed if it is not empty. Each error message created is tagged with an error code, placing it in an internal category of errors depending on the portion of the system to which it relates. Table 3.3 defines the help file identifier ranges used in the encoding of the error codes and provides a brief description of each range's message.

The Help system contained within the *CS1 Sandbox* is essentially a HyperText Markup Language (HTML)-based [91] display system. Individual help files were authored using a simple text editor and categorized with a unique identification number which corresponded to a matching error code. For example, an error message explaining the restriction of a language component is encoded with an help file identifier value in the sixteen to twenty range.

Once generated, error message are displayed to the user in the lower portion of the *CS1 Sandbox* environment in a scrollable/selectable list. By clicking on an error in the list, the user can prompt the environment to scroll the source code display to the line the error was located on, highlight the offending line of source code, and enable a button on the interface to provide additional information about the error.

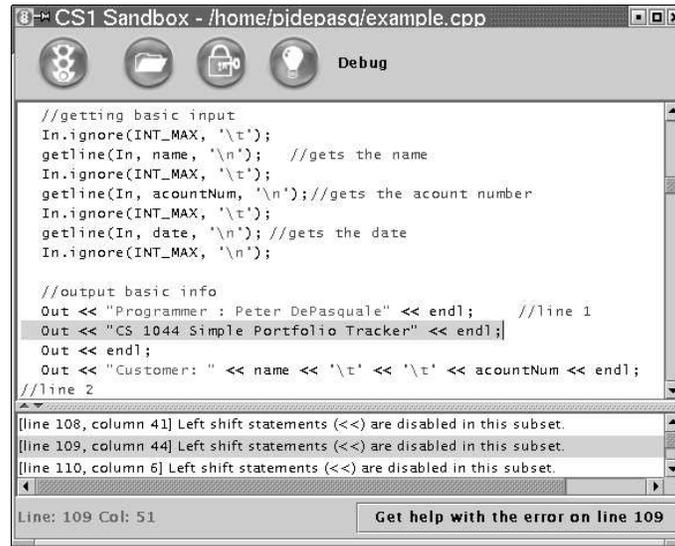


Figure 3.9: The *CS1 Sandbox* Programming Environment showing several errors following an interpretation attempt. Note that one error has been selected by the user, and that the offending line has been highlighted in the source code text area. Additionally, the “Get help with the error on line...” button has appeared, tailored to the selected error.

When possible, error messages are reused to present similar information customized to a particular situation. To accomplish this errors and warnings originating from the underlying components (lexer, parser, etc.) are parsed for specific pieces of data. Once obtained, these data segments are then re-inserted into the actual message, in a more friendly and novice-oriented display. For example, in the case of the user attempting to include a header file which happens to be disabled (e.g. `#include <math>`), the name of the header file is parsed from the statement and re-inserted into an appropriate error message. This message, “*The %s header file is disabled in this subset.*”, is then presented to the user with the name of the header file replacing the `%s` token of the message.

By clicking the “Get help with the error on line X” button in the lower portion of the interface, the user requested that additional help be displayed explaining the selected error message. This action triggers the loading and displaying of the corresponding help file in a separate help window (see Figure 3.10).

Each help file utilizes HyperText Markup Language (HTML) as the underlying source, and applies Cascading Style Sheets (CSS) to assist in providing a consistent look-and-feel to the help contents. Each help file provides a brief description of the error, a list of common causes of the error, tips to avoid the error (geared toward novice programmers), and additional code examples where the error is elaborated and a solution provided. Keywords in each help file are linked to a glossary page for further reference by the user so that a user could quickly lookup a definition, if needed.

3.2.5 Deployment of the Environment

A great deal of consideration was given to how the *CS1 Sandbox* programming environment was to be deployed to the subjects during the dissertation experiment. Knowing that there was a high probability of

Table 3.3: Help File Identifier Ranges and Descriptions

Error code range	Description of topic
0	Default error code (unknown error)
1–5	Reserved error codes
6–10	ANTLR lexer error codes
11–15	ANTLR parser error codes
16–20	Restriction (subset) error codes
401–500	ANTLR syntax analysis error codes
501–510	ANTLR interpretation error or warning codes

finding errors in the environment (or needing to make some other type of modification) while approximately one hundred and twenty subjects were using it, the experimenter realized that subsequent releases of the software were very likely going to take place during the course of the experiment. This, in fact, was the case as such changes as additional help files, corrections to the interpreter, GUI modifications, and adjustments to error messages providing additional information.

At first, it was considered that a selection of compact discs (or some other type of inexpensive removable media) would be made available for the subjects to borrow – for installing the software to their personal machines. Not only would this implementation be costly in terms of discs, it relied on the subjects returning the discs to the experimenter so that other subjects could borrow them for their own installations. Additionally, the solution did not consider the three different closed-laboratory locations that would need to be upgraded with the new software. With roughly one hundred and twenty subjects, this process was deemed impractical.

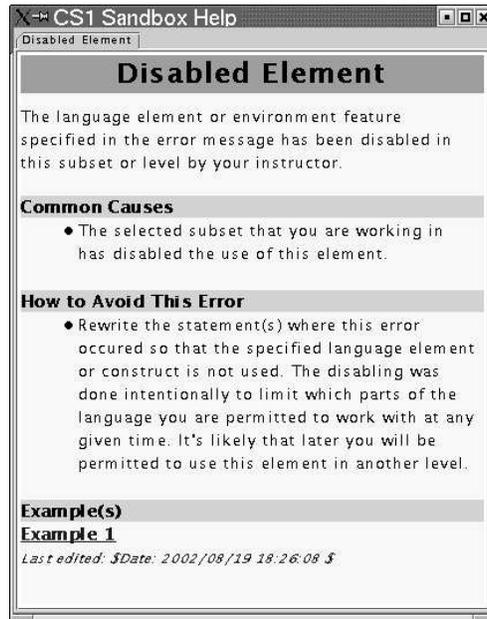
The next solution considered was to make the software available for download via the File Transfer Protocol (FTP) from a central server. Access to a server was not an issue as the data collection server (see Section 3.4.2) could double as a FTP server for this purpose. However, this put the onus on the subjects to download and install any upgrades, and did nothing to ease the requirement of upgrading the software on the roughly ninety closed-laboratory machines (many of which the experimenter would not have administrator-level access to perform such upgrades).

The Java Web Start [76] technology was selected to solve all of the above problems with the least amount of hassle to both the experimenter and the subjects. Java Web Start is an application that assists in the deployment of Java applications. Once it resides on a computer, it can automatically check for newer releases of a Java application (such as the *CS1 Sandbox*).

When a revised version is detected on a server (in this case the data collection server), the application (or just the modified portion of the application) is downloaded and then installed, thereby updating the existing application. By installing Java Web Start in each of the closed-laboratories, and by requiring the subjects to install it on their personal machines, it was ensured that whenever the subjects attempted to launch the *CS1 Sandbox* the most recent version was available for them.

Java Web Start is designed to attempt to upgrade the software automatically whenever the application in question is initially executed. However, if the computer's network connection is disabled, or the server containing the updates is unavailable, the currently installed version is executed in its place.

By using Java Web Start, the experimenter was able to provide updates in a timely fashion without a great deal of effort. End-users also reported that upgrades were swift and simple to accomplish – just start the application and wait a few moments for the update to be downloaded.

Figure 3.10: A Sample *CS1 Sandbox* Help Window

3.3 Deployment of *CS1 Sandbox* Subset Information (server)

In order to provide the configuration and control of the language subsets to the subject's client programming environments, a second application was created. This application allowed for the design and deployment of the subset control information in a standardized format. This section details this application, known as the Subset Creation Tool.

3.3.1 Subset Creation Tool

The subset creation tool was designed with two key goals in mind. The first was the ability to easily create, modify, and save subset creation files. The second goal was to then selectively provide one or more of these files upon request to a subject's copy of the *CS1 Sandbox* programming environment.

Initially, a prototype of the Subset Creation Tool was developed and tested in conjunction with a project assignment in a section of CS6724 (End User Programming) during the fall of 2001. An evaluation of the application was performed as a deliverable in the course, the participants comprising likely CS1 instructors drawn from the Computer Science Department's faculty and graduate student population. The preliminary prototype used in the study attempted to represent the language components (from which the subsets are created) in a hierarchical manner in an attempt to demonstrate the notion of subsets building from each other. This hierarchical method was universally disliked by the instructors participating in the study for its complexity and lack of clarity with regard to how the subsets were constructed.

Based on this feedback, the prototype was redesigned to display the language components in a categorical manner. See Figure 3.11 for a screen capture of the final version of the Subset Creation Tool's interface used to create a subset configuration.

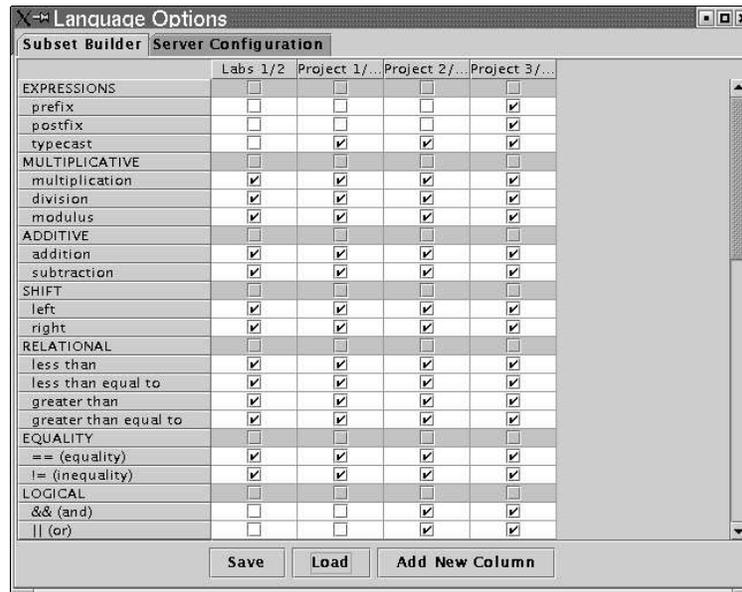


Figure 3.11: The Subset Creation Tool’s interface used to create language subsets

Using this interface, an instructor can specify one or more subsets for deployment to the programming environment. Each subset (columns in the spreadsheet-like interface) can comprise one or more language components. For each language component, a check mark in a subset column indicates that the component is enabled within the environment when the subset is in use.

Following the creation of the subsets, an instructor can save the configuration information to a file for later editing or serving to the programming environments used by the students. To facilitate ease of data transfer with the smallest footprint possible, the configuration file was encoded using the eXtensible Markup Language (XML) format. This file (internally known as a subset configuration file) is download by the student and applied to the programming environment in order to apply the subsets to the environment (see the discussion of this process in Section 3.3.2). A sample of the XML file created by the subset creation tool is included in Appendix H.

The second feature of the Subset Creation Tool was the ability to selectively make subset configuration files available to connecting clients. To do this, a second interface was provided such that one or more simultaneous subset files could be provided at any given time³.

The server configuration portion of the Subset Creation Tool provided the ability to select one or more subset configuration files to be available for downloading, a toggle for starting and stopping the server, easy access to controlling on which communication port the server will listen for connections, and the ability to maintain a persistent list of server configuration settings for ease-of-startup. A screen capture of the server configuration interface is shown in Figure 3.12.

Using this interface, the instructor can add a subset configuration file to the file of available files provided by the server (by using the “Add Subset File” button). Files added by the instructor are then listed in the large area to the left of the button. Currently Figure 3.12 is displaying one file added to the server’s list of

³Despite this capability, only one subset configuration file was made available to subjects participating in the dissertation’s experiment. The ability to serve multiple files was thought to be an asset to instructors who may use the software in the future to support multiple sections of an introductory programming course.

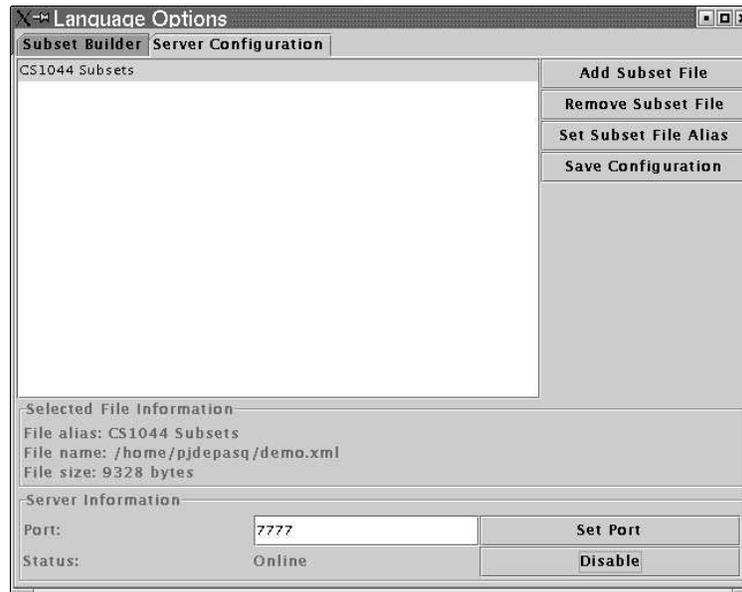


Figure 3.12: The Subset Creation Tool’s interface used to distribute subset configuration files. The large white region is used to display and select subset configuration files available to be served to connecting *CS1 Sandbox* clients (programming environments). In the figure above, a single subset configuration file (CS1044 Subsets) has been added to the list of available files and selected by the user. Once selected, information about the file is displayed in the region labeled Selected File Information.

available files (labeled CS1044 Subsets). This subset configuration file was given a name alias by the instructor by using the “Set Subset File Alias” button. File aliases were provided to allow instructors the ability to craft a more appropriate text string shown to *CS1 Sandbox* users when selecting a subset to download (as opposed to the name of a subset configuration file; e.g. “cs1044.xml”).

Other controls on the interface support the removal of subset configuration files from the list of available files provided by the server (“Remove Subset File” button), and the ability to save the current server configuration (“Save Configuration” button). If a subset configuration file in the list of available files is selected by the instructor, a section of information is updated detailing the file’s alias (if applicable), the true name and location of the subset configuration file (in the example in Figure 3.12 the selected file is named `/home/pjdepasq/demo.xml`), and the size of the file in bytes.

Finally, the communications port on which the server is listening for communications connections can be configured in the lower region of the interface. Additionally, the server can be toggled enabled (on-line) or disabled (off-line) if desired by the instructor (by using the “Enable” or “Disable” toggle button – currently shown as a “Disable” button).

The server configuration functionality was not examined during the usability study performed in conjunction with CS6724 mentioned earlier (see Section 3.3.1), as the server software was not fully designed or developed at the time of the study. However, the server software (both configuration interface and communications code) was able to support the experiment undertaken by this dissertation.

3.3.2 Downloading and Applying Subset Information

Language subsets must be applied to the *CS1 Sandbox* programming environment by end-users by first downloading a subset configuration file from a server. To accomplish this, the user selects the *Download Language Subsets...* menu item via the “Subset button” (see Section 3.2.3), displaying a configuration panel for attempting to connect to the server. This panel is displayed in Figure 3.13.



Figure 3.13: The interface used to select from which server and port to attempt a subset download



Figure 3.14: The interface used to select which subset configuration file to download

For the purposes of the dissertation’s experiment, the hostname and port number of the subset server was coded into the panel by default, removing a configuration step for the users. However, this information can be easily modified allowing the user to attempt to connect to any machine and port on the Internet.

Once a connection has been established to the server, the panel is replaced with one that permits the user to select which subset configuration file to download to their programming environment. In Figure 3.14, the user has selected the subset configuration file appropriate for the experiment.

Following the download of a subset configuration file, a menu appears in the upper right area of the environment, permitting the user to select which subset is applied. Following the selection of a specific subset from the list (obtained via the subset configuration file) by the user, the configuration of each programming language component is to be applied in the environment’s system (thereby enabling or disabling the component). This process is detailed in Section 3.3.3. Once a subset has been applied to the programming environment, the user can then proceed in their editing, compiling, debugging, and execution activities.

A subset configuration display window was developed to allow the students to view the current status of the subset which was applied to their environment. A sample of this window is shown in Figure 3.15. Next to each language element a small status indicator dot is shown. If the dot appeared filled with a red color, use of that element was disabled. A dot filled with a green color indicated an enabled language element.

3.3.3 Enforcement of Subsets

Following the download of a subset configuration file into the programming environment and a subset selected by a subject, a Restriction software object is created to manage the current state of the language components. Each language component item (e.g. character data types, *switch* statements, *for* statements, and the like) is represented in the Restriction object with a simple boolean value.

If the stored boolean value associated with a given language component is the representation of the value “*true*”, then the component is considered to be “on” or enabled, and the student is permitted to use it. Conversely, a component whose associated value is “*false*” is disabled and the student is prohibited from using it (while this level is in use).

During each compilation or execution attempt, both the parser and the semantic validator verifies the status of a language component (enabled or disabled). For example, if during the parsing pass a *switch*

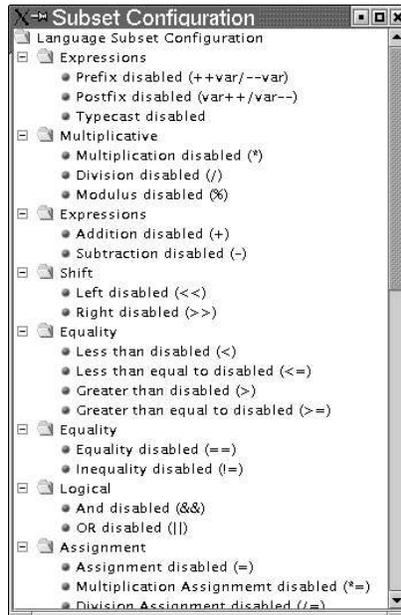


Figure 3.15: The interface used to display the current status of language constructs/elements. (The small dot to the left of the element is a color status indicator.)

statement was recognized, the parser would check with the Restriction object to test the status of the *switch* statement. If the Restriction object returned an enabled state (“*true*” value), the parser would continue with its work.

If, however, the Restriction object returned a disabled state (“*false*” value), the parser would then create a suitable error message indicating to the user that the switch statements were disabled in the current subset and that the user was not able to use them. The parser would continue by attempting to parse the remainder of the source code file.

Subjects using the *CS1 Sandbox* with a subset applied have little recourse upon seeing such an error message but to remove the offending statement from their program. There is no mechanism in the programming environment to disable the application of a subset once it is enabled (in order to attempt to bypass the error), other than the ability to select another subset.

Those subjects required to use *CS1 Sandbox* with the application of subsets used a version of the software with a single parameter set such that by default all language components were disabled until a subset configuration file was downloaded and a subset selected by the subject. In this fashion, it was reasonably assured that no subject was able to circumvent the subset enforcement techniques⁴.

⁴There was no mechanism to ensure that a subject did not first author their program (outside of the closed-laboratory setting) in the .NET programming environment and then migrate/move their code to the *CS1 Sandbox* environment when some of all of the bugs were successfully removed. Additionally, no practical method of preventing this type of behavior in an experiment of this scope and length was found.

3.4 Data Collection Mechanism

As important as the internal workings of the environment was to a successful deployment among student-subjects, the data collection was equally important to successfully capturing the student's performance with the environment. Thus, an invisible (to the subject) data collection portion was enclosed in the programming environment to capture and send data from the student's computer (either in the closed-laboratories or elsewhere for the out-of-laboratory programming projects) to a centralized server. To support such a need, two additional software components needed to be designed, implemented, and tested for use – a data collection client and server. The client component would be embedded in the *CS1 Sandbox* – facilitating the ability to collect data and send it to the data collection server (residing on a separate machine). These components are described in the sections that follow.

3.4.1 Client-side

Within the *CS1 Sandbox* programming environment, a small software component collected the following data from the subject's platform each time the subject attempted to compile/interpret their program.

1. PID - the Virginia Tech Personal Identifier, used to trace each subject's data
2. PID validated - a flag indicating if the PID was validated successfully via a prompting dialog
3. Date / time - a timestamp of the execution of the subject's program
4. Platform name - the operating system the subject is using (used for debugging and troubleshooting purposes)
5. Machine name - the Internet Protocol (IP) address and hostname of the machine the subject is using (used for debugging and troubleshooting purposes)
6. Java version - the version of the Java Runtime Environment (JRE) which the subject is using (used for debugging and troubleshooting purposes)
7. Sandbox version - the version of the *CS1 Sandbox* that the subject is using (used for debugging and troubleshooting purposes)
8. Source code listing - a complete copy of the program source code being compiled/executed
9. Error listing - a listing of any syntax or semantic errors in the program source code being compiled/executed
10. Abstract Semantic Tree (AST) - the ANTLR AST representation of the parsed program source code being compiled/executed

This data was collected each time the subject attempted to compile or execute the source code contained in the environment. Using Java threads and socket communications, the programming environment would silently attempt to open a connection to the data collection server following the completion of each compilation/execution attempt and submit the collected data for storage.

If the connection to the server failed (due to network connectivity problems, the server was down, etc.) the data was not sent and the environment continued processing without any indication to the student that anything had gone awry. This implementation was designed in this fashion for several reasons:

Firstly, without delaying the experiment, it was not possible to create more advanced fail-safe mechanism which would attempt at a later time to submit the data collected. Secondly, in keeping with the notion that the environment should be as simple as possible, it was decided that notifying the students that the data collection portion had failed might just confuse the average student, and possibly alarm those students who had no previous programming experience.

3.4.2 Server-side

The server used to collect the subject's data for this experiment was based on a Linux operating system (Mandrake 8.1 [48]) running the MySQL [55] database server, and the Apache [79] web server with the PHP [78] scripting language. Working in concert, these applications and servers received the incoming data from the subjects' programming environments (client-side), stored it in temporary files and submitted it to the database for long-term storage.

Data contained in a submission to the server was packaged as five sections of a stream of bytes separated by a sub-stream of consecutive equal signs. The submission took the form shown in Figure 3.16.

```

Data collection section - contains information about
the subject's system, IP address, subject identifier
=====
Source code section - contains the subject's source
code submission verbatim
=====
Error section - lists syntax and semantic errors (if any)
=====
Output section - lists output produced to the cin and
cout streams
=====
Subset information - lists the subset configuration
details for the programming environment at the time
of the submission
=====

```

Figure 3.16: Data Submission Format

Once received, the PHP scripting language was used to receive the data via a communications socket, and place the information transferred to a temporary file on the system. An example of the data transferred (and the corresponding temporary file) is shown in Figure 3.17.

Once the file was saved, the PHP process disassembled the contents of the data file into distinct pieces and sent them to the MySQL database via a connection to the database's server process. The database server in turn inserted the data as a new record in the database. In this fashion, each file that arrived on the server was stored in the database (and also stored in a file). Upon successful completion of the database insertion, the temporary file was moved to an archival location, thereby preserving the originally transferred information in the event of a database failure. The entire system is represented pictorially in Figure 3.18.

On occasion, subjects would enter a series of equal signs in the comments of their source code files. These added comments would ((unbeknown to the user) disrupt the use of such marks as section separators in the temporary data file, and would in turn halt the addition of the file to the database (other incoming data files were not affected). In these cases, the temporary files remained in a holding area, where they could be added to the experiment data set by hand.

```

System: x86:Windows 2000:5.0
IP: lark (198.82.184.113)
Java: Sun Microsystems Inc.:1.3.1
Sandbox: 1.2.2.F
User: ***removed***
PID validated: false
=====
//Program Author: ***removed***
//Date of submission: September 4, 2002
//Lab Section: Wednesday 1:25PM
//
#include <fstream>
using namespace std;
int main () {
    //declare the output stream and open the file.
    ofstream outFile;
    outFile.open("lab1out.dat");

    //Initial preamble for output
    outFile << "Welcome to cs1044 - Fall 2002" << endl;
    outFile << endl << endl;

    //Student specific data output
    outFile << "My name is: " << "***removed***" << endl;
    outFile << "I am in the " << endl;
    outFile << "1:25PM section of 1044." << endl;
    outFile << "I am using the following programming environment: " << endl;
    outFile << "Sandbox" << endl;

    //Close the output file
    outFile.close();

    //Return from function
    return 0;
}
=====
Errors: none
-----
Output:

=====
Subset information:
none
=====

```

Figure 3.17: A sample data submission from a subject's programming environment

3.5 Debugging Support (for the *CS1 Sandbox* Developers)

Two features of the programming environment were developed exclusively for the purpose of supporting the development team in the creation of the *CS1 Sandbox*. These included an internal debugging window and an Abstract Syntax Tree viewpane. Both of these features are discussed in the following sections.

3.5.1 Debugging Window

During the development of the *CS1 Sandbox*, it became necessary to display debugging messages originating from ANTLR's lexer, parser, semantic checker, and interpreter interfaces. Thus, the developers created a debugging window used to display these messages.

In addition to ANTLR-generated messages, Java error messages were being sent to the standard error stream during execution of the programming environment. However, it was noticed during testing that when the *CS1 Sandbox* was executed via Java's Web Start (in the same fashion a subject would be using

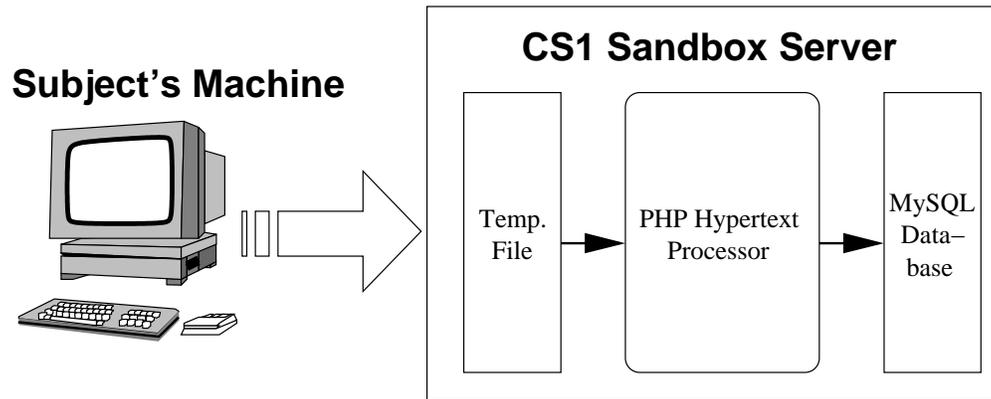


Figure 3.18: A pictorial representation of the entire *CS1 Sandbox* data collection system

the environment), such messages would not be displayed as there was no command shell in which to display them. Generally, these messages took the form of Java run-time exceptions and programmer-added debugging statements. To handle this situation, the development team added an additional display to the Debugging Window to display the Java-specific error messages (e.g. Java exceptions).

```

-----
errorCode = 16
type = level
lineNumber = 75
columnNumber = 2
endLineNumber = -1
endColumnNumber = -1
errorMessage = [line 75, column 2] Constant identifiers are disabled in this subset.
rawErrorMessage = Constant identifiers
-----
errorCode = 16
type = level
lineNumber = 76
columnNumber = 2
endLineNumber = -1
endColumnNumber = -1
errorMessage = [line 76, column 2] Constant identifiers are disabled in this subset.
rawErrorMessage = Constant identifiers
-----
errorCode = 16
type = level
lineNumber = 77
columnNumber = 2

```

Figure 3.19: The internal debugging interface

The final representation of the debugger's display window is shown in Figure 3.19. Three text panes (separated by tabbed controls) help separate the content of the lexer/parser/semantic checker messages, the interpreter messages, and messages generated to the error stream.

3.5.2 AST Viewer

In addition to the debugging output window, a second display was created to present the Abstract Syntax Tree (AST) resulting as the output of the ANTLR parser. The display (see Figure 3.20) represents the tree structure in a interface the user can manipulate. That is, any parent node can be expanded or collapsed by a simple mouse click. In this fashion, the AST can be manipulated to help the developer view a specific area on which they wish to concentrate.

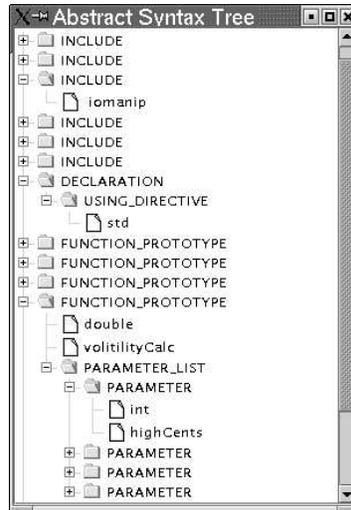


Figure 3.20: The Abstract Syntax Tree (AST) display interface

The actual tree representation produced is accomplished through a call to the ANTLR Java libraries. In addition to containing the Java code to create and manipulate the AST, they also contain functionality to display an AST textually or graphically. Following the completion of the parsing of the program source code by ANTLR, the library returns an AST representation in a tree-like data structure. Given this structure, a call to an ANTLR library routine produces the graphical representation of the tree in a user interface component which can be embedded into a window frame (as was achieved in the *CS1 Sandbox*).

The AST viewer became a resource of tremendous importance during the development of the *CS1 Sandbox*, as it permitted developers to analyze exactly how the statements in a given program were parsed by ANTLR. Additionally, with each execution of the parser, additional AST view windows could be created and were persistent in the environment. This led to the development team modifying the input source code, reparsing the input, and comparing the resulting AST trees in order to see the effect of the changes upon the resulting AST. This mechanism greatly assisted the development team in validating the correctness of the ASTs produced by ANTLR and the *CS1 Sandbox* source code.

Chapter 4

The Design of the Experiment

4.1 The Purpose of the Experiment

This experiment sought to capture a range of data to determining the effect of the application of a simplified interface and language subsets to the programming environment that novice programming students use in completion of a CS1 course. To accomplish this, the experiment utilized the *CS1 Sandbox* and .NET programming environments in order to capture a variety of data from participating subjects.

In addition to the electronic data captured through the on-line system, surveys, closed-laboratory assignments, out-of-laboratory project scores, and focus group feedback were collected about the subjects. This chapter elaborates on the design and issues surrounding the experiment undertaken by this dissertation.

4.2 The Classroom Environment

In the Fall, the experiment of this dissertation was conducted in conjunction with the two Computer Science major's-only sections of CS1044 (Introduction to Programming in C) [81] at Virginia Tech. Both of the lecture sections were taught by William McQuain (instructor in the Computer Science department), and contained a total of 168 registered students. A comprehensive breakdown of the characteristics of the population of the course is shown in Table 4.1.

Table 4.1: CS1044 Section Breakdown, Fall 2002

	Course Section		Total
	8:00 am	10:10 am	
Students Enrolled	79 ¹	89 ²	168
Students Participating in the Experiment			
Male	73	79	152
Female	4	9	13

¹Two students from the 8:00 am section opted out of participation of the experiment.

²One student from the 10:10 am section was dropped from the experiment due to having participated the previous semester in the course.

Each section met at their respective lecture times on Monday and Wednesday mornings throughout the semester. In addition, students were required to attend each week one of seven closed-laboratory sessions (see Section 4.2.1 for a discussion of the laboratories) supervised by this author (four sessions), or by Ines Khelifi (three sessions). In addition, each closed-laboratory session included and an undergraduate teaching assistant.

During the semester, each student enrolled in the course was required to complete two written tests, five out-of-laboratory programming projects (see Section 4.2.2 for a discussion of the projects), and a final examination. Table 4.2 enumerates the individual weights assigned to each deliverable in the course and Table 4.3 provides the course schedule of topics covered. Appendix C contains a copy of the course syllabus distributed to all enrolled students.

Table 4.2: Grade Weightings for CS1044

Item	Weight
Closed-laboratory assignments (13)	20%
Out-of-laboratory projects (5)	
Project #1	4%
Project #2	6%
Project #3	8%
Project #4	10%
Project #5	12%
Tests (2)	
Test #1	8%
Test #2	12%
Final Exam	20%
Total	100%

4.2.1 Closed Labs

The two-hour laboratories associated with CS1044 ran for a period of fourteen weeks, with presentations regarding an introduction to the rules and structure of the laboratories constituting the first laboratory. Laboratory sessions were held on Wednesdays (two), Thursdays (three), and Fridays (two). During each laboratory, the students were given a programming project to complete and submit for a score. As noted in Table 4.2, the scores of these closed-laboratory sessions comprised 20% of each students' grade in the course.

Each closed-laboratory programming project built directly on concepts covered during the lecture sections from that week, and was designed to be completed by the average student in 90 minutes (well before the end of the two hour laboratory). A summary of each laboratory assignment is provided in Table 4.4.

Table 4.3: Schedule of topic coverage for CS1044

Week #	Lecture Topic
1	Administration & Policies Problem Solving Programming Process & Program Design
2	Program development
3	C++ Syntax Fundamentals
4	C++ Input/Output Basics
5	Booleans & Selection
6	Iteration
7	Iteration Functions
8	Functions
9	Arrays
10	Arrays User-defined Types
11	Structured Types
12	Structured Types Searching
13	Sorting
14	<code>string</code> operations

Table 4.4: Summary of CS1044 Closed-Laboratory Assignments

Laboratory number	Concepts Targeted	Laboratory Summary
1 (Sept. 4-6)	Initial compilation	This laboratory included several simple fill-in-the-blank style questions (for the student to answer on their laboratory assignment sheet) from the course notepack and was primarily designed to force students to bring their laboratory books to class. In addition, learning a basic familiarity with their programming environment was the goal of the session. Finally, this assignment required the students to submit their completed programs to the automated grading system (discussed later). (see Appendix D.1)
2 (Sept. 11-13)	Expression completion, monetary representation, expression assimilation	Students were asked to complete several mathematical expression statements, and assimilate the expressions into a partially completed program which was provided. (see Appendix D.2)

continued on next page

Table 4.4: Summary of CS1044 Closed-Laboratory Assignments (*continued*)

Laboratory number	Concepts Targeted	Laboratory Summary
3 (Sept. 18-20)	Writing first program from scratch, output formatting, problem/solution formulation	Students were asked to author their first complete (in-laboratory) program. Program involved processing output formatting monetary values as integers, and required specific output formatting. (see Appendix D.3)
4 (Sept. 25-27)	Boolean programming logic	First portion of the laboratory was completion of simple textual-logic (fill-in-the-blank) problems. The programming portion involved identification of a Chinese New Year animal by authoring an appropriate calculation using the <code>mod</code> operation and boolean logic tests. (see Appendix D.4)
5 (Oct. 2- 4)	Conditionals (switch or if statements) and input loop processing	The students were required to read input from a file and count the number of vowels, whitespace and “other” characters and output a short table. (see Appendix D.5)
6 (Oct. 9-11)	Nested loops, sentinel loop control	Students were required to selectively print the N^{th} word from an input line. The program was to terminate upon recognition of a sentinel value. (see Appendix D.6)
7 (Oct. 16-18)	Applying a single user-defined function	An adaptation of laboratory #6; except that students were required to apply a single user-defined specified function to the program. (see Appendix D.7)
8 (Oct. 23-25)	Applying multiple user-defined functions	Calculate the area of a rectangle given its coordinates, utilizing two user-defined functions including pass-by-reference, and a return-typed function. (see Appendix D.8)
9 (Oct. 30-Nov. 1)	Array bounds checking	Deals with array bounds checking without implementing an actual array by having the students conceptualize an array and its contents via a list. Students print which cells are out of bounds, invalid (bad index number), used (contains a value), or unused (is currently empty). (see Appendix D.9)
10 (Nov. 6-8)	Array creation and manipulation	Populate an array with characters based on input file commands, and print the contents of specific locations based on additional input commands (<code>set/show</code>). Students must also be able to identify invalid locations (bad indices), or attempting to show an uninitialized location in the array. (see Appendix D.10)

continued on next page

Table 4.4: Summary of CS1044 Closed-Laboratory Assignments (*continued*)

Laboratory number	Concepts Targeted	Laboratory Summary
11 (Nov. 13-15)	Structures	Read in a collection of data (movie name, year of release, movie rating, and name of lead actor) and place it in a structure designed to hold such information. Manipulate the contents of one field in the structure and print the results. (see Appendix D.11)
12 (Nov. 20-22)	Arrays of structures, searching	Read in a collection of data (movie name, year of release, movie rating, and name of lead actor) and place it in an array of structures. Print only those items which are explicitly identified by commands in the input file. (see Appendix D.12)
13 (Dec. 4-6)	Sorting	An adaptation of laboratory #12; read in the data and produce a sorted list of the movies stored in the structure according to ascending alphabetical order. (see Appendix D.13)

4.2.2 Programming Projects

Over the course of the semester, the students were given five programming projects to take home and complete on their own (without help from other students). Table 4.5 briefly summarizes the schedule of projects for the semester and provides a short description of each.

Each project was scored automatically as it was submitted to the Department of Computer Science Curator [50] software, which assigns a numeric score for the required output. Each Curator score was out of 100 points (except for the last project which was based on 200 points). Following projects 1, 3, 4, and 5, a software style analysis³ was performed by the various undergraduate teaching assistants and Ines Khelifi, and then applied to the student’s submission possibly resulting in points being deducted from their Curator provided project score.

Generally speaking, students were given approximately two class weeks to complete the projects, but varied due to such factors as the course timetable, success in the closed-laboratory sessions, Thanksgiving break, and the scheduling of the subsequent programming project.

4.3 Approval of Experiment Participation

In accordance with the Virginia Tech policy on experiments involving human subjects [83], and Institutional Review Board (IRB) application (see Appendix A) was submitted to the department chair

³From the course syllabus – “A number of the out-of-lab programming projects will also be graded for adherence to good software engineering principals, including documentation, design, conformance to the stated specification, and programming style. Each project specification will include explicit guidelines that you will be expected to follow. The TAs will grade your (first) submission to the Curator that received the highest score, and e-mail you the results.” (see Appendix C)

Table 4.5: Summary of CS1044 Out-of-Laboratory (Take-Home) Programming Projects

Project number	Release date of project specification (or draft)	Due date of project	Drop dead date for project	Summary of project taken from specification
1	Sept. 10	Sept. 30	Oct. 4	Write a program that will read simple data for a single (financial) stock and produce a report. This will require reading a data from an input file, doing simple calculations, and writing a simple output file. In addition, the program will have to use a C++ selection mechanism to make decisions regarding several possible charges. (see Appendix E.1)
2	Oct. 7	Oct. 15	Oct. 21	Modify and extend your implementation for Project 1 to employ an input-failure/sentinel loop and more sophisticated calculations. The format of input and output files will be radically different. (see Appendix E.2)
3	Oct. 21	Oct. 29	Nov. 4	Modify and extend the implementation for Project 2 to accommodate a design based upon a good procedural decomposition of the assigned problem. The input file will have precisely the same format as for the previous project. The format of output file will be slightly modified to reflect some additional requirements. (see Appendix E.3)
4	Oct. 29	Nov. 12	Dec. 6	Write a program that is capable of both encrypting text and decrypting text, applying a given permutation to sequences of characters taken from the original text. (Required the use of user-defined functions and arrays.) (see Appendix E.4)
5	Nov. 13	Dec. 9	Dec. 12	Modify and extend your implementation for Project 3 to take advantage of arrays and structures to improve the organization of the data and to support additional operations. You will also explore searching and sorting arrays of structures. (see Appendix E.5)

(Dennis Kafura) and then to the Virginia Tech IRB for approval. Both entities approved the design of the experiment. When the course commenced, the student-subjects were given a description of the experiment and a informed consent form (contained in the IRB application) to complete and return to the

experimenter. This was completed on the first day of class (and on demand for those students who did not attend the first day of class). Since the informed consent form doubled as a first-day attendance sheet, it was assured that this author obtained a clear indication of intent to participate or request for non-participation for each student. Students who did not attend class on the first day, were dealt with on a case-by-case basis by this author.

Approximately 195 students registered on the first day of the course. Students who dropped the course during the semester were removed from consideration in the experiment. Of the 168 students enrolled in the class at the completion of the experiment, only two explicitly requested at the start of the experiment that their data not be used, while one additional student was removed from participation for having previously taken the course, leaving 165 participating student-subjects.

All of the students removed from consideration in the experiment but remaining in the course sections were subject to the same workload and deliverables for the class, including use of the *CS1 Sandbox*, if applicable based on their selection of their closed-laboratory section - as per William McQuain's directive.

4.4 Subject Demographics

The subjects for this experiment were drawn from those Computer Science major students enrolled in the CS1044 in the Fall of 2002. However, the experiment did not comprise the entire freshman class, as some students were placed in CS1704 (Introduction to Data Structures and Software Engineering, the second course in programming) because they had previously tested out of CS1044, or passed the Advanced Placement (AP) test with a sufficiently high result. Additionally, some freshmen were placed (by the Computer Science department's academic advising staff) in CS1114 (Introduction to Problem Solving and Computing) due to lack of prior programming experience/education and low mathematics aptitude scores.

Students enrolled in the CS1044 course sections involved in the experiment were limited to Computer Science major students only. Three additional sections of CS1044 were taught during the semester – those containing non-major students (Business Information Technology, for example) and those students who had declared Computer Science to be their academic minor, but these were not included in this experiment.

To provide a basis of comparison for the entire freshman class (regardless of initial course placement), a demographic survey (see Appendix B for a copy of the survey questions) was conducted through the two sections of CS1104 (Introduction to Computer Science; essentially a CS0 course in the ACM/IEEE-CS curriculum guidelines) which all major students must take during their freshman year, including some transferring into the Computer Science program from other institutions. This survey permitted this author to capture demographic statistics (age, gender, prior programming experience, college entrance exam scores) on the CS1044 population (used later in the analysis of the experiment) as well as similar values for a sample of the entire freshman class.

The survey was administered to 213 students in CS1104 on a volunteer basis. Data collected by the survey was also correlated (by keying the survey to a respondent's student identification number) to those subjects participating in the experiment (where we also had access to their student identification number as participants in the lecture and laboratory sections). Department records indicate that the Fall semester saw 287 freshman and 39 transfer students enroll in the Computer Science program. Thus the survey represents a 65.5% saturation (188 of 287) of the freshman class, and 30.8% (12 of 39) of the transfer student population⁴. It is speculated that not all of the transfer students were required to enroll in CS1104, thus the significantly lower percentage surveyed.

⁴Twelve sophomores and one junior students also participated in the survey.

Table 4.6 shows the demographic distribution of the experiment's subjects across both lecture section while Table 4.7 compares the key demographic elements surveyed for freshmen in the experiment against freshman survey respondents. Note that the freshmen participating in the experiment have a highly similar composition to their peers in the entire freshmen class.

One point worth elaborating on is the prior programming experience scores reported in both tables. Each survey respondent was asked to rank their previous programming experience on a scale from 1 (no experience) to 7 (had worked full time) with each of a 11 application development, web development, and miscellaneous (programming) languages. These were chosen by this author as likely languages that a semi-experienced high school student may have been exposed to (either formally or self-taught). As such, the possible range of result for each student would be 0 (no programming experience whatsoever) to 77 (a professional-level programmer in every language – with the option to write in additional languages and experience ratings).

Table 4.6: Demographic Distribution of the Experiment's Subjects

	8:00 am lecture section	10:10 am lecture section	Total (Percent)
Gender			
Male	73	79	152 (92.1%)
Female	4	9	13 (7.9%)
Academic Level			
Freshman	73	80	153 (92.7%)
Sophomore	4	3	7 (4.2%)
Transfer	0	5	5 (3.0%)
Age			
17	12	9	21 (12.7%)
18	50	64	114 (69.1%)
19	10	11	21 (12.7%)
20	2	3	5 (3.0%)
21+	3	1	4 (2.4%)
Exiting high school?			
Yes	70	83	153 (92.7%)
No	7	5	12 (7.2%)
Entrance exam means			
SAT	1277.7	1275.9	1276.7
ACT	27.3	29.1	28.5
Prior Programming Experience Score			
Mean	8.6	8.6	8.6
Median	8.0	8.0	8.0

4.5 Division of Subjects into Experiment Groups

In order to compare the effect of the *CS1 Sandbox* programming environment's simplified interface and application of subsets, the subjects were logically divided into three distinct groups based upon their selection of closed-laboratories as follows:

Table 4.7: Demographic Distribution and Means of Freshmen (Experiment Subjects and Survey Respondents)

	Freshmen Only	
	Experiment Subjects	Survey Respondents
Gender		
Male	141 (92.2%)	174 (92.5%)
Female	12 (7.8%)	14 (7.4%)
Age		
17	21 (13.7%)	24 (12.8%)
18	113 (73.9%)	140 (74.5%)
19	15 (9.8%)	19 (10.1%)
20	3 (2.0%)	4 (2.1%)
21+	1 (0.7%)	1 (0.5%)
Entrance exam means		
SAT	1279.0	1277.7
ACT	28.6	28.3
Programming Experience Score		
Mean	8.9	9.0
Median	8.0	8.0

1. subjects using Microsoft's Visual C++ .NET programming environment⁶,
2. subjects using the *CS1 Sandbox* programming environment without the use of language subsets, and
3. subjects using the *CS1 Sandbox* programming environment with the application of language subsets.

Table 4.8 elucidates the distribution of the environments among the specific laboratories.

Each experiment group had certain restrictions set upon it (primarily which programming environment to use throughout the course of the semester). These restrictions are described in detail in the next section.

Table 4.8: Distribution of Subjects and Programming Environments

Laboratory #	Day / Time of Laboratory	# of Subjects	Programming Environment Used
1	Wednesdays - 1:25 to 3:15pm	25	<i>CS1 Sandbox</i> without subsets
2	Wednesdays - 4:00 to 5:45pm	27	Microsoft .NET
3	Thursdays - 11:00 to 1:00pm	21	<i>CS1 Sandbox</i> without subsets
4	Thursdays - 5:00 to 6:50pm	22	<i>CS1 Sandbox</i> with subsets
5	Thursdays - 7:00 to 8:50pm	27	Microsoft .NET
6	Fridays - 10:10am to 12:00pm	21	<i>CS1 Sandbox</i> with subsets
7	Fridays - 1:25 to 3:15pm	25	Microsoft .NET

⁶.NET is the current standard programming environment all freshmen students were required to use in their first and second year courses.

4.5.1 Programming Environment Usage Requirements

Subjects using the *CS1 Sandbox* groups were instructed to use only the *CS1 Sandbox* programming environment for the first nine closed-laboratories and the first three out-of-laboratory projects, then moving to the .NET programming environment for the remaining work. The duration of use for the *CS1 Sandbox* environment was a two-fold decision. Firstly, the environment was not capable of supporting several advanced features required during the later stages of the course. These features included arrays, structures, and enumerated types. Secondly, by migrating the subjects back to the standard programming environment (.NET), it would be ensured that each subject was a capable user of the .NET programming environment before proceeding on in their curriculum, but it permitted the collection of additional important data - how the students would react to a professional strength environment (.NET).

Subjects who used the .NET programming environment were instructed to use .NET for the entire semester's work. Both groups were asked not to experiment with other programming environments, specifically .NET for *CS1 Sandbox* users and *CS1 Sandbox* for .NET users. For both environments, a tutorial (printed in the case of .NET, and on-line in the case of the *CS1 Sandbox*) was provided to get started using the programming environment, as well as a live tutorial was performed during the first week of the closed-laboratories.

Subjects who used the *CS1 Sandbox* with subsets were required (prior to attempting a compilation or execution of their source code each time the programming environment was used) to download the language subset configuration information to their programming environment and apply a specific subset. No subset-caching capabilities were built into the *CS1 Sandbox* programming environment - resulting in the subjects being forced to download their subset configuration file with each execution of the programming environment (and prior to attempting to compile any source code they had entered). The process of downloading and applying a subset is covered in Section 3.3.2.

During the course of the semester, four language subsets were released to the subjects. Table 4.9 elaborates the sequence and contents of the subsets, as well as the course deliverables to which they mapped (which subsets were used with each closed-laboratory and out-of-laboratory programming projects).

A special "Migrating to .NET" help session (run by this author) was offered during the weekend of the switch from the *CS1 Sandbox* to .NET (November 3rd) for those students interested in attending. Approximately twenty five of the eighty seven subjects who used the *CS1 Sandbox* attended the help session. The remaining students were strongly encouraged to utilize the existing .NET tutorial (contained in their course note pack) for assistance in migrating to the .NET environment. It is believed that some subjects had prior experience with Microsoft Visual C++ (highly similar to .NET) or .NET itself. Data collected in support of the out-of-laboratory programming projects indicate that a handful of students may have utilized the .NET programming environment outside the confines of the laboratories, as these subjects had extremely low (less than five) compilation attempts in completing their programming projects - where the mean number of compilations were more than 100.

4.6 Data Collection

Three sets of data collection were employed during the course of this experiment. The following sections elaborate on each of the types of data collected and provide details on their method of collection.

Table 4.9: *CS1 Sandbox* Subset Release Sequence

Subset #	Deliverable(s)	Language Items Enabled
1	Laboratories 1–2	Multiplicative: *, \, % Additive: +, - Shift: <<, >> Relational: <, <=, >, >= Equality: ==, != Assignment: = Declarations: boolean, char, short, int, long, double Jump: return Libraries: climits, cmath, fstream, iostream
2	Project #1 & Laboratories 3–4	(All from subset #1 plus the following) Expressions: typeid Assignment: *=, \=, +=, %=, -= Declarations: const Selection: if, if-else, switch Jump: break Libraries: cstdlib, iomanip, string
3	Project #2 & Laboratories 5–7	(All from subset #2 plus the following) Logic: &&, Iteration: do-while, while, for
4	Project #3 & Laboratories 8–9	(All from subset #3 plus the following) Expressions: prefix, postfix Iteration: for statements with loop control variable declarations

4.6.1 Electronic Data (Closed-Laboratory and Out-of-Laboratory Programming Projects)

All subjects who used the *CS1 Sandbox* programming environment (both with and without subsets) were subjected to continual data collection during use.⁷ This data collection process was contained in a separate process as part of the *CS1 Sandbox* programming environment which sent the following data to a database server each time the subjects attempted to compile or execute their programs:

1. PID - the Virginia Tech Personal Identifier, used to key each subject with their experiment data
2. Date / time - a timestamp of the compilation/execution attempt
3. Platform name - the operating system the subject is using (used for debugging and troubleshooting purposes)

⁷The subjects were made aware of this activity via their experiment acceptance agreement and were reminded verbally by this author in the lecture sections and closed-laboratories.

4. Machine name - the Internet Protocol (IP) address and hostname of the machine the subject is using (used for debugging and troubleshooting purposes)
5. Java version - the version of the Java Runtime Environment (JRE) which the subject is using (used for debugging and troubleshooting purposes)
6. Sandbox version - the version of the *CS1 Sandbox* that the subject is using (used for debugging and troubleshooting purposes)
7. Source code listing - a complete copy of the source code being compiled/executed
8. Error listing - a listing of any syntax or semantic errors in the code being compiled/executed
9. Abstract Semantic Tree - the ANTLR AST representation of the parsed code being compiled/executed

In order to capture a subject's PID, the subject was prompted to enter their PID and corresponding university password (for validation of the PID) via a prompting message on first use of the environment on any given computer. During the prompting for the PID, the subject was reminded of the data logging and the project's need to associate the captured data with their individual identities for the purpose of a successful experiment.

On occasion, a few subjects incorrectly inserted their Social Security number as their PID, especially during the first few weeks of use with the *CS1 Sandbox* environment. In order to correctly correlate this captured data with the correct student, the date/time and machine name values were used to reconstruct which subject had made the submission.

This was particularly the case for the closed-laboratory submissions, where attendance records and instructor knowledge of which machine the subject would commonly use when in the laboratory would help with the identification process. Thus, nearly all of the collected data was correlated with the appropriate subjects. In one closed-laboratory instance a subject was found to be submitting data via another subject's PID while the second subject was also submitting data with their PID. Once recognized, knowledge of which machine the first subject utilized on a weekly basis assisted in correctly identifying which data was submitted by each subject in question.

Data logging of experiment data was achieved in the background while using the *CS1 Sandbox*, without disrupting the student's use of the environment. With few (known) exceptions, nearly every compilation/execution was captured by the database logging system. In four instances during closed-laboratories (of nearly 775 instances of students participating in a closed-laboratory), no data was recorded for a particular subject, despite the subject working in the environment and receiving a score for the laboratory period. These lapses were attributed to network connectivity problems in the laboratory, rather than database logging problems (as other submissions during the same period were accomplished).

The data logged by this process resulted in over 45,000 database records for the 87 participating students during the nine weeks of the closed-laboratories and three out-of-laboratory programming projects. Electronic data collection was not possible for those subjects using the .NET programming environment as there was no way to modify the environment to support the sending of data with each compilation/execution attempt to the data collection server.

4.6.2 Academic Scores

As with the programming work, the subjects knowingly had their numeric scores recorded for the purposes of this experiment. This encompassed the scored results of the closed-laboratories, out-of-laboratory

programming projects, two mid-term tests, and the final exam. All of the academic performance results (scores) were also correlated back to each individual subject, and thus the experimenter could compare all aspects of a subject's performance (e.g. a subject's project grade could be compared to the number of compilation attempts required to complete the project).

4.6.3 Post-Project Data Collection / Surveys

Following each of the five out-of-laboratory programming projects assigned during the semester, the subjects were asked to complete both a short on-line survey (see Appendix F.1) and submit an electronic data sheet via the Curator system. The survey queried the subjects on their impressions on working with their programming environment (both .NET and the *CS1 Sandbox*), the project itself, the estimated number of compilation and execution attempts required to complete the project (used to capture this information for the subjects who used the .NET programming environment, since automatic data collection was not possible), where the subjects obtained assistance in completing the project, and the like.

The data collection sheet served as a back-up mechanism for the estimated number of compilations and executions. It was speculated that by completing the sheet as the subjects worked on the project, a truer representation of the number of compilations/executions would be reported, though anecdotally one subject admitted to creating false numbers feeling that the requirement of tracking the compilations/executions took up too much of their time. It was not determined if their claims were true or to what direction the fabrications diverged from the actual data – as the subject had used the .NET programming environment for the entire semester.

4.6.4 Error Recognition Experiment and Focus Group

Following the completion of use of the *CS1 Sandbox*, a focus group was organized to solicit comments about the use of the subjects' programming environments. Participation in the focus group was completely voluntary, though in exchange for a subject's participation free pizza and sodas were offered.

Announcements were made in laboratory sections and on the course web-based message discussion board soliciting the subject population for participants in the focus group, however the focus group's size was limited to 45 subjects – 15 from each of the experiment group categories (subjects who used the .NET programming environment, subjects who used the *CS1 Sandbox* without subsets, and subjects who used the *CS1 Sandbox* with subsets). In all 29 subjects participated; 11 subjects who used .NET, 7 who used the *CS1 Sandbox* without subsets, and 11 who used the *CS1 Sandbox* with subsets.

The focus group meeting took place one Monday evening in a campus lecture room after all *CS1 Sandbox* activities had ceased (use of the *CS1 Sandbox* in out-of-laboratory programming assignments and in closed-laboratory activities). The event started with the participants taking a short test experiment designed to examine their capability of recognizing syntax and semantic errors (or lack thereof) in small programs or program fragments (see Section 5.3).

Following a short food break, the subjects were queried in an open-forum setting as to the strengths and weaknesses of the programming environment they had been using. Subjects were recorded using a digital audio recorder, and asked to specify their programming environment before making comments.

This page intentionally left blank.

Chapter 5

The Results of the Experiments

5.1 Closed-Laboratory Data Collected

As mentioned in Section 4.6, the closed-laboratory portion of the experiment yielded several data sets that this dissertation wished to examine in addition to the common data for each submission. These sets included both raw (collected) and derived data as follows:

- the raw scores from the closed-laboratory assignments,
- the total number of compilation/execution attempts (derived), and
- the number of errors generated by compilation/execution attempts (derived).

Of these three data sets, only the first (closed-laboratory scores) included those subjects using the .NET programming environment in the closed-laboratory sessions. The last two sets (number of compilations and number of errors) were only obtainable when using the *CS1 Sandbox* programming environment as .NET had no mechanism for data capture of the number of compilations and/or errors generated.

Following the completion of the ninth closed-laboratory and third out-of-laboratory programming project, those subjects using the *CS1 Sandbox* environment (both without and with subsets) were migrated to the .NET programming environment. All subjects then used the .NET programming environment to complete the remaining four closed-laboratories (laboratories #10 through #13) as well as the two remaining out-of-laboratory projects. During this phase of the experiment, only the closed-laboratory scores were able to be collected from the data.

Closed-laboratory programming submissions were scored automatically by the Computer Science department's Curator system which scores each submission based on the correctness of the output produced by the submission:

“The Curator compares your output file to the correct output file, line by line. Each line is broken into ”tokens” consisting of characters separated by whitespace (blanks or tabs). The tokens from each line of your output file are compared to those from the corresponding line of the correct output file; your score for each line is determined by the number of correct tokens you produce and the number of points assigned to that line [by the instructor].” [49]

5.1.1 Closed-Laboratory Raw Scores

During the closed-laboratory portion of the experiment data sets were collected from a total of 2,122 laboratory solutions (and all work leading up to a solution) over a period of 13 weeks involving all of the 165 participating subjects. The 2,122 value is the number of laboratory assignments which were scored across the entire semester, but is slightly less than the maximum possible number of laboratory assignments due to students missing laboratories which they were not permitted to make up.

Table 5.1 shows a weekly breakdown of the mean laboratory scores according to the experiment groups (.NET, *CS1 Sandbox* without subsets, and *CS1 Sandbox* with subsets). Corresponding Figure 5.1 shows the mean closed-laboratory scores collected each week of the experiment broken down by the programming environment in use. Following week 9, subjects using the *CS1 Sandbox* migrated to using the .NET programming environment (see Section 4.5.1). However, for the purposes of post-migration analysis the graph reflects the original programming environment they were tasked to use.

Table 5.1: Mean Closed-Laboratory Score Details (laboratories #1 through #13). Note that following laboratory #9 all *CS1 Sandbox* subjects were migrated to .NET, however for data analysis purposes the experiment grouping remained intact. All scores collected from submissions to Curator grading system.

Laboratory #	Mean laboratory scores		
	.NET	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets
1	99.37	98.30	98.93
2	97.65	97.80	98.49
3	94.91	92.77	92.28
4	97.24	97.68	97.80
5	99.81	99.84	99.24
6	100.00	96.82	96.83
7	100.00	99.84	100.00
8	100.00	100.00	100.00
9	100.00	99.52	100.00
10	97.66	94.14	94.53
11	99.62	99.19	100.00
12	88.53	92.14	92.41
13	99.12	99.28	98.61

5.1.2 Number of Compilation / Execution Attempts

Yet another area under examination by this dissertation is the number of compilation/execution attempts. As noted earlier (refer to Section 3.4), each time a subject attempted to perform an attempt at compilation (parsing and semantic validation under the *CS1 Sandbox*) or execution (interpretation) a data log was submitted to the data collection server. In doing so during the course of the experiment, 8,997 such records were collected from subjects using the *CS1 Sandbox* without subsets during closed-laboratory sessions.

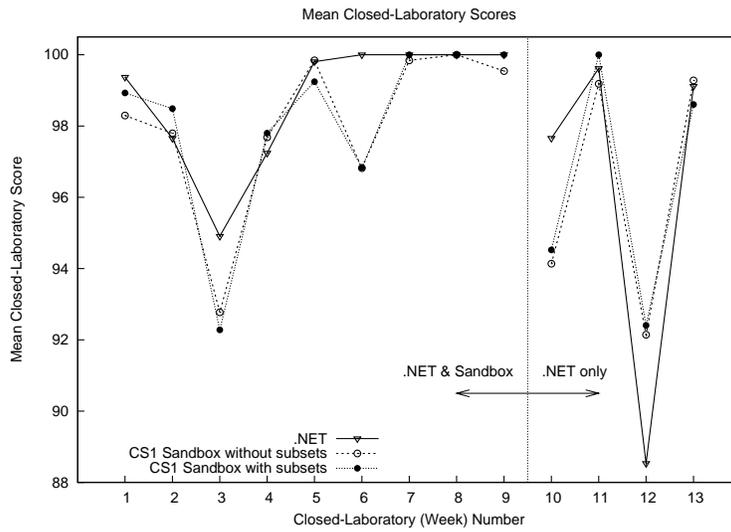


Figure 5.1: Graph of mean closed-laboratory scores (laboratories #1 – #13)

Similarly, 7,233 records were collected from subjects using the *CS1 Sandbox* with subsets applied. These 16,230 records account for roughly 36% of the more than 45,000 total data records collected during the experiment.

Table 5.2 provides the detailed values of each of the mean number of compilations per subject across all of the closed-laboratory sessions for both *CS1 Sandbox* experimental groups. Figure 5.2 graphs this data pictorially. It should be noted that it was not possible to collect such data from the users of the .NET programming environment and *CS1 Sandbox* users after they have migrated to the .NET programming environment following closed-laboratory #9.

Table 5.2: Mean Number of Closed-Laboratory Compilation / Execution Attempts per Subject (laboratories #1 through #9). Note that following laboratory #9 all *CS1 Sandbox* subjects were migrated to .NET for which data collection was not possible.

Laboratory #	Mean number of laboratory compilations	
	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets
1	3.21	5.60
2	20.61	20.45
3	45.48	35.28
4	27.86	24.54
5	20.09	15.63
6	32.21	24.05
7	18.68	14.93
8	23.00	16.12
9	14.93	13.81

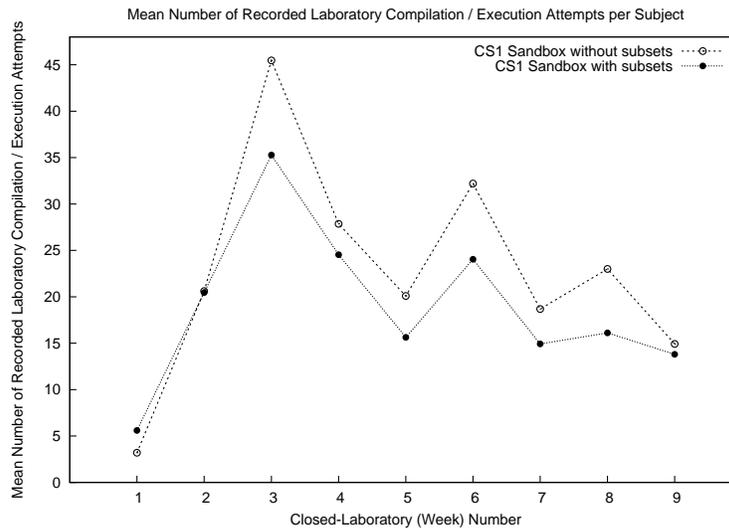


Figure 5.2: Graph of mean number of closed-laboratory compilations (laboratories #1 – #9)

5.1.3 Number of Errors Produced

In addition to the collection of the number of compilations and scores assigned for the closed-laboratory programming assignments, the experiment also obtained a total count of the number of errors (e.g. lexer/parser errors, semantic errors, and run-time/interpretation errors) which were recorded during the data collection process. It should be reiterated that the data presented in this data set is derived from the total number of errors each subject incurred for all compilation/execution attempts in solving the closed-laboratory programming task(s) during each laboratory.

Table 5.3 elaborates the mean number of errors per student during each of the closed-laboratory sessions, while Figure 5.3 graphs this data.

Table 5.3: Mean Number of Closed-Laboratory Compilation Errors per Subject (laboratories #1 through #9). Note that following laboratory #9 all *CS1 Sandbox* subjects were migrated to .NET for which data collection was not possible.

Laboratory #	Mean number of laboratory compilation errors	
	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets
1	1.59	6.93
2	14.34	22.26
3	72.18	52.86
4	45.80	56.49
5	18.16	29.29
6	21.35	21.38
7	30.50	25.30
8	58.91	40.02
9	25.67	20.02

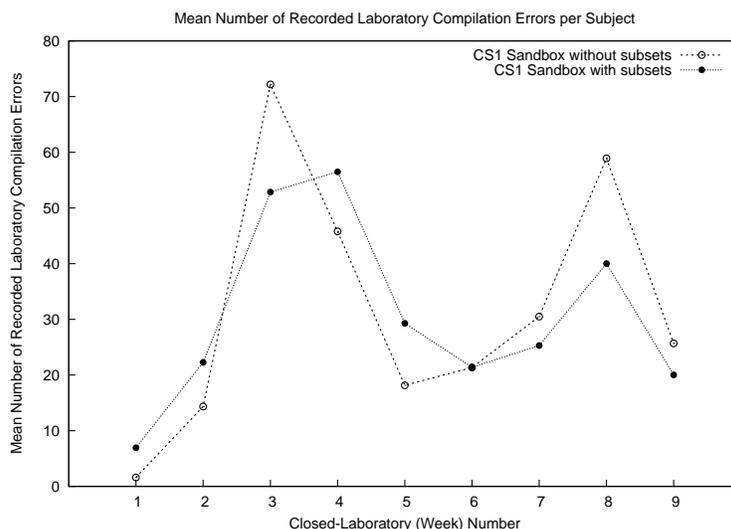


Figure 5.3: Graph of mean number of closed-laboratory compilation errors (laboratories #1 – #9)

5.2 Out-of-Laboratory (Take-Home) Project Data Collected

Similar to the closed-laboratory portion of the experiment, out-of-laboratory project data was collected while the subjects were completing their out-of-laboratory programming assignments. The data sets collected during this phase of the experiment included both raw and derived data as follows:

- the raw scores from the out-of-laboratory programming projects,
- the number of compilation/execution attempts (derived),
- the number of errors generated by compilation/execution attempts (derived), and
- the total logged time-on-task in completing the projects (derived).

Of these four data sets, only the first (mean scores) could include those subjects using the .NET programming environment. The last three sets were only obtainable when using the *CS1 Sandbox* programming environment (as .NET had no mechanism for data capture of the number of compilations and/or errors generated).

Each subject in the experiment was required to complete a total of five out-of-laboratory (take-home) programming projects. These projects (see Appendices E.1, E.2, and E.3) were presented via the course web site, and varied in allowable completion time from nine to twenty seven days (refer to Table 4.5). Each subsequent project grew in complexity as material covered in the lectures was incorporated into the projects. Project submissions were also scored automatically by the Computer Science department's Curator system.

The results used in this analysis are the highest raw score from the Curator for each subject. In order to establish a grade for each subject, the raw score was modified to apply a late penalty or early completion bonus, (which varied with each project). The modified score was then returned to the subject, and the

highest modified score was recorded as their Curator-derived score for the project. The modified score was used as the basis of the course assigned score for the project. However, only the initial raw score was used in this experiment.

Post-Curator scoring (hand-scoring for adherence to the specification – comments, use of data types, formatting, etc.) was also applied for several of the projects (#1, #3, #4, and #5), but was not taken into account during this analysis. Subjects who did not submit a project (and earned a zero score) were not counted in the analysis, however any subject who submitted a solution to the Curator (for scoring) was included, regardless of the score received.

In the presentation and analysis of the data collected from this portion of the experiment, a number of subjects' results were removed from consideration for violating the conditions of the experiment. During the entire experiment subjects were asked not to use any other programming environment except for the one assigned to them. Via the post-project surveys, this author determined that number of subjects had admitted to violating this directive. Thus, it was deemed that subjects who had admitted (in their post-project survey) to using another programming environment were removed from the resulting data set for each project in which they used another programming environment. Additionally, subjects who did not complete the post-project surveys were also removed, since it could not be clearly discerned if they too had tainted their results by using another programming environment.

The total breakdown of the number of subjects who completed each out-of-laboratory programming assignment is reflected in Table 5.4.

Table 5.4: Number of subjects completing each out-of-laboratory project

Environment used (total eligible subjects)	Programming Project #				
	# 1	# 2	# 3	# 4	# 5
.NET (78)	70	71	72	59	51
<i>CS1 Sandbox</i> without subsets (44)	38	39	34	33	31
<i>CS1 Sandbox</i> with subsets (43)	38	38	39	32	35

5.2.1 Project Raw Scores

This portion of the experiment collected the raw scored results from the submission of subjects' out-of-laboratory programming project solutions to the Curator.¹ The maximum recorded score for each

¹Recall that a student has multiple opportunities for each programming project to submit “correct” solutions. As such, students may “run out of submissions” prior to completing a project.

subject earned through the scoring of their submissions to the Curator is used as both the data set collected and as the basis of their score for the assignment. The score reported to the student is then potentially adjusted with an early bonus, late penalty, or deductions from hand-scoring by the teaching assistants. However, only the raw score is considered herein.

Table 5.5 provides a breakdown of the mean out-of-laboratory project scores for all 5 take-home programming projects, while Figure 5.4 shows a corresponding graph of the data.

Table 5.5: Mean Out-of-Laboratory Project Score Details (projects #1 through #5). Note that following project #3 all *CS1 Sandbox* subjects were migrated to .NET, however for data analysis purposes the experiment grouping remained intact.

Project #	Mean project scores		
	.NET	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets
1	97.14	98.39	96.95
2	98.35	98.05	96.79
3	98.56	98.85	94.79
4	95.97	98.15	95.69
5	97.50	97.34	97.11

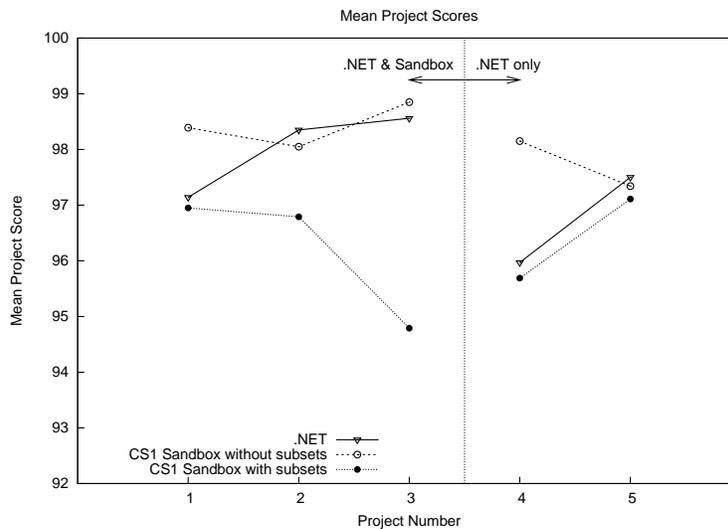


Figure 5.4: Graph of mean out-of-laboratory project scores (projects #1 – #5)

5.2.2 Number of Compilation / Execution Attempts

Another data set under examination by this dissertation is the number of compilation/execution attempts performed by each subject during the completion of the out-of-laboratory programming projects. As noted earlier (refer to Section 3.4), each time a subject attempted to perform an attempt at compilation (parsing and semantic validation under the *CS1 Sandbox*) or execution (interpretation), a data log was submitted to the data collection server. 14,992 such data records were collected from subjects using the *CS1 Sandbox* without subsets during closed-laboratory sessions. Similarly, 11,638 records were collected from subjects using the *CS1 Sandbox* with subsets applied.

Table 5.6 provides the detailed values of the mean number of compilations across all of the out-of-laboratory projects for both *CS1 Sandbox* experimental groups². Figure 5.5 shows the graph of each mean. It should be noted that it was not possible to extract such data from the users of the .NET programming environment and the *CS1 Sandbox* users who had migrated to the .NET programming environment following project #3.

Table 5.6: Mean Number of Out-Of-Laboratory Compilation / Execution Attempts per Subject (projects #1 through #3). Note that following project #3 all *CS1 Sandbox* subjects were migrated to .NET for which data collection was not possible.

Project #	Mean number of project compilations	
	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets
1	147.53	132.95
2	108.13	95.03
3	95.94	64.56

²Recall that these values exclude subjects who did not complete the post-project surveys or admitted in the survey to using another programming environment than the one assigned.

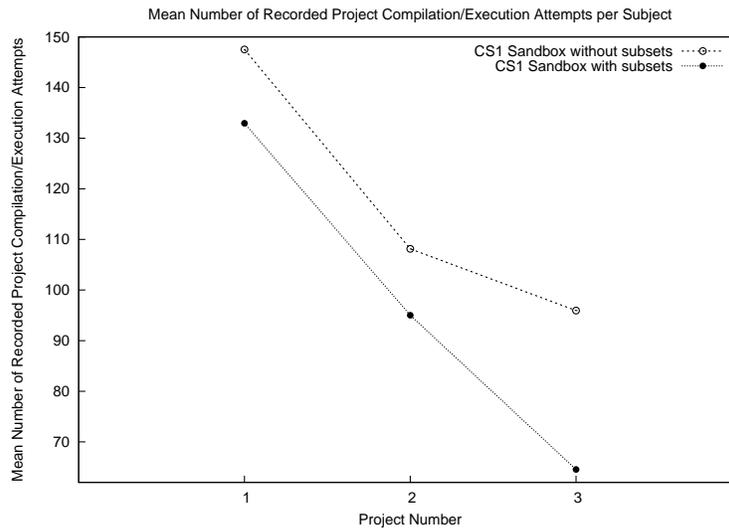


Figure 5.5: Graph of mean number of out-of-laboratory project compilations (projects #1 – #3)

5.2.3 Number of Errors Produced

In addition to the collection of the number of compilations and scores assigned for the out-of-laboratory programming assignments, the experiment also extracted a count of the number of errors (e.g. lexer/parser errors, semantic errors, and run-time/interpretation errors) which were recorded during the data collection process.

Table 5.7 elaborates the mean number of errors per subject during each of the closed-laboratory sessions, while Figure 5.6 graphs the graphed data sets. The data presented below is comprised of the following:

- *CS1 Sandbox* without subsets – the mean number of syntactic, semantic, and interpretation errors for all users in this experiment grouping.
- *CS1 Sandbox* with subsets – the mean number of syntactic, semantic, and interpretation errors for all users in this experiment grouping. This excludes any errors produced as the result of the application of language subsets (subset errors).

By removing the subset errors from consideration, we can compare the mean number of actual errors generated by both experiment groups. Without the removal of this superfluous data, the mean number of errors are artificially inflated for the users of the *CS1 Sandbox* with subsets programming environment.

Table 5.7: Mean Out-Of-Laboratory Compilation Error Details (projects #1 through #3). Note that following project #3 all *CS1 Sandbox* subjects were migrated to .NET for which data collection was not possible.

Project #	Mean number of project compilation errors	
	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets
1	105.32	92.55
2	45.79	55.66
3	189.38	228.51

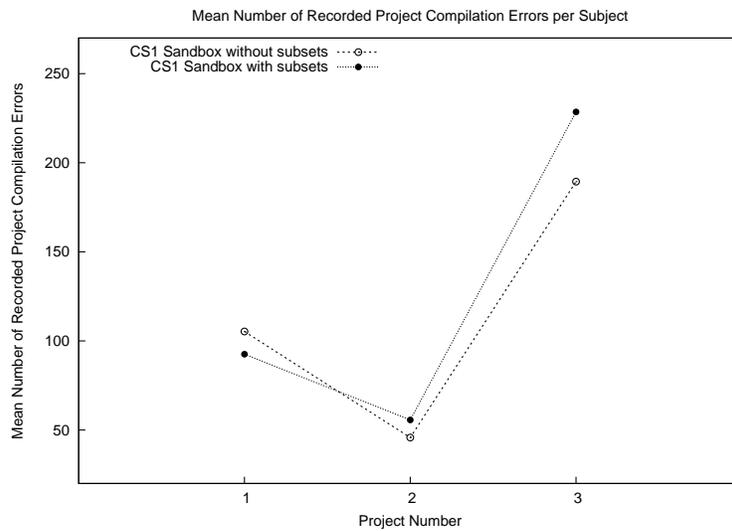


Figure 5.6: Graph of mean number of out-of-laboratory project compilation errors (projects #1 – #3)

5.2.4 Overall Time-on-Task

Time-on-task values for each out-of-laboratory project were computed by analyzing the first and last recorded compilation/execution attempts using the *CS1 Sandbox* programming environment. For each subject, a time-on-task value was calculated – the result of subtracting the time³ of the first recorded compilation/execution attempt from the last recorded attempt. The resulting difference was identified as the subject’s time-on-task for completion of the project. Without directly monitoring the subject’s work activities (impossible for an experiment of this scale), this method provides only an approximation of the time-on-task and includes down-time as the subject’s activities spanned multiple days (for instance while the subject slept or was attending classes). Table 5.8 provides the details of the calculated means, while Figure 5.7 graphs the resulting mean time-on-task values across all three out-of-laboratory programming projects.

Table 5.8: Mean Out-Of-Laboratory Time-On-Task for Programming Projects (projects #1 through #3). Note that following project #3 all *CS1 Sandbox* subjects were migrated to .NET for which data collection was not possible.

Project #	Mean project time-on-task (in seconds)			
	<i>CS1 Sandbox</i> without subsets	% of total time allowed	<i>CS1 Sandbox</i> with subsets	% of total time allowed
1	135:59:06 (489546.03)	24%	151:43:51 (546230.95)	26%
2	65:13:41 (234820.59)	19%	62:58:41 (226721.42)	19%
3	59:17:50 (213470.38)	18%	54:19:26 (195566.18)	16%

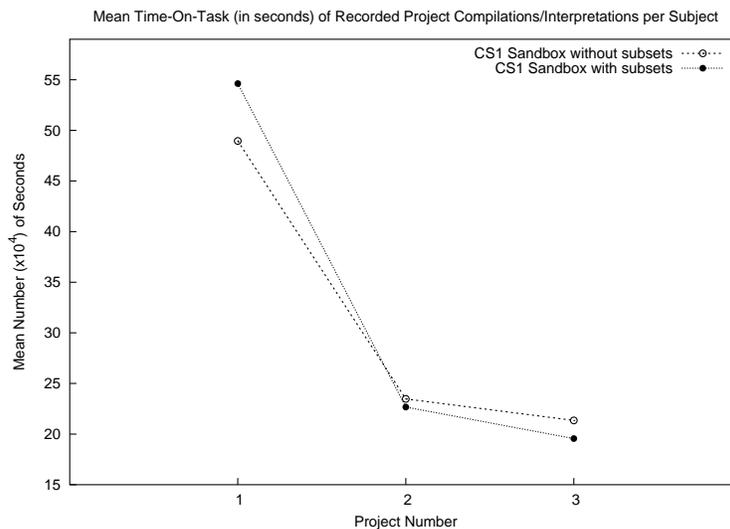


Figure 5.7: Graph of mean time-on-task (in seconds) of out-of-laboratory projects (projects #1 – #3)

³The recorded time was converted to a UNIX timestamp; the number of seconds since January 1st, 1970 GMT.

5.2.5 Post-Project Survey Results

As discussed in Section 4.6.3, following each of the five out-of-laboratory projects the subjects were asked to complete a post-project survey. Each survey was conducted on-line and subjects entered their responses via a web-based survey system provided by Virginia Tech [84].

The five surveys consisted of 13 common questions (see Appendix F), though the first three surveys had one or two additional questions which were later removed⁴ As part of the survey, subjects were required to enter their PID to identify their responses for each survey, and to be able to collate their responses to their experiment group.

As the survey results were tabulated, this author maintained a list of which subjects had completed the surveys and e-mailed reminders to the subjects to do so. Initially, the surveys were treated as a partial deliverable for the out-of-laboratory projects (in an effort to spur a high response rate of the surveys), however in the final tabulation of course grades by William McQuain it was decided to drop the surveys as a requirement for project delivery. The survey results were modified by this author to only correct errant PID entries where subjects had incorrectly entered their Social Security Number as their PID. Using course enrollment information, the correct value was entered for the subject. Several entries were submitted without a PID or Social Security Number and thus could never be associated to a particular subject or experiment group.

In the presentation and analysis of the data collected from this portion of the experiment, a number of subjects' results were removed from consideration for violating the conditions of the experiment. During the entire experiment subjects were asked not to use any other programming environment except for the one assigned to them. Via the post-project surveys, this author determined that number of subjects had admitted to violating this directive. Thus, it was deemed that subjects who had admitted (in their post-project survey) to using another programming environment were to be removed from the resulting data set for each project in which they used another programming environment.

5.2.5.1 Self-Reported Data

Three questions in each of the post-project surveys asked each subject to self-report the following to the best of their ability:

- the approximate number of days each subject worked on the out-of-laboratory project (see question #7 of the sample survey in Appendix F),
- the approximate number of compilations required to complete the out-of-laboratory project (see question #8 of the sample survey in Appendix F), and
- the approximate number of executions required to complete the out-of-laboratory project (see question #9 of the sample survey in Appendix F).

The resulting data obtained from these survey questions is presented in Tables 5.9 and 5.10⁵. Figures 5.8 and 5.9 graph the mean values presented in these tables.

⁴These removed questions asked the students which programming environment they had been using (which was redundant as we knew this from the rest of the experiment), and if they had previously used a visual programming environment (as opposed to command-line based) before their enrollment at Virginia Tech.

⁵Since the *CS1 Sandbox*'s programming environment did not differentiate between the notion of compilation and execution (successful compilations automatically initiated an execution attempt), and the .NET programming environment contains such differentiation (by the use of two separate menu items or toolbar buttons), the data between these two reported categories was summed to establish a more consistent count of the perceived task(s).

Table 5.9: Mean self-reported values from the post-project survey question: *How many days did you work on the project?* (Note: only count a day once when you worked on the project one or more times)

Post-Project Survey #	Mean reported days on task		
	.NET	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets
#1	2.78	3.53	3.11
#2	2.29	2.44	2.11
#3	2.48	2.21	2.13
#4	3.38	2.91	2.97
#5	4.04	4.83	3.60

5.2.5.2 Rating Data

To facilitate data analysis of the collected data from the surveys, each subject's response in rating-type questions (such as *How would you categorize your experience in starting a new project/program in the programming environment you used?* (*Very Easy, Easy, Medium, Difficult, Very Difficult*)) were converted to a numerical value as follows: Very Easy=1; Easy=2; Medium=3; Difficult=4; Very Difficult=5.

A copy of the common questions in the survey has been reproduced in Appendix F. Results from the survey which are examined by this dissertation are presented in the Tables 5.11 through 5.14. Corresponding graphs of the mean responses for each experiment group are shown in Figures 5.10 through 5.13.

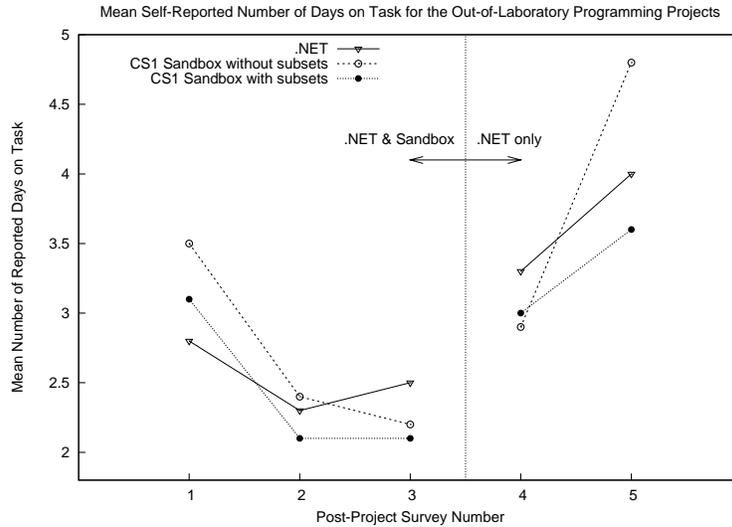


Figure 5.8: Graph of mean self-reported number of days on task for the out-of-laboratory programming projects

Table 5.10: Mean self-reported values from the post-project survey questions: *How many attempts at compilation did you perform while completing the project? (please quantify, don't enter "some")* and *How many attempts at execution did you perform in completing the project? (please quantify, don't enter "some")*

Post-Project Survey #	Mean reported # of compilations & executions		
	.NET	CS1 Sandbox without subsets	CS1 Sandbox with subsets
#1	76.3	66.6	46.8
#2	69.4	73.7	68.1
#3	63.0	53.7	57.5
#4	136.0	115.6	129.6
#5	172.2	160.5	156.2

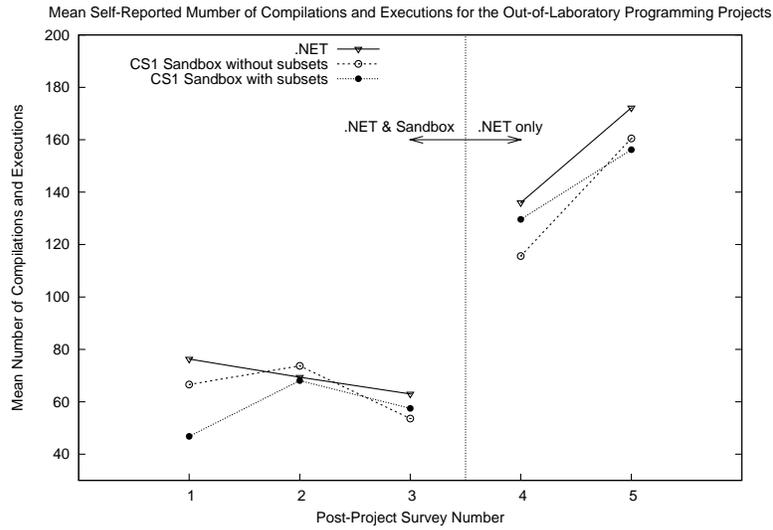


Figure 5.9: Graph of mean self-reported number of compilation and execution attempts for the out-of-laboratory programming projects

Table 5.11: Mean rating values from the post-project survey question: *How would you categorize your experience in starting a new project/program in the programming environment you used? (Very Easy=1, Easy, Medium, Difficult, Very Difficult=5)*

	Post-Project Survey				
	#1	#2	#3	#4	#5
.NET	2.0	1.6	1.8	1.7	1.7
CS1 Sandbox without subsets	1.8	1.8	2.0	2.5	2.2
CS1 Sandbox with subsets	2.0	1.8	1.9	2.9	2.5

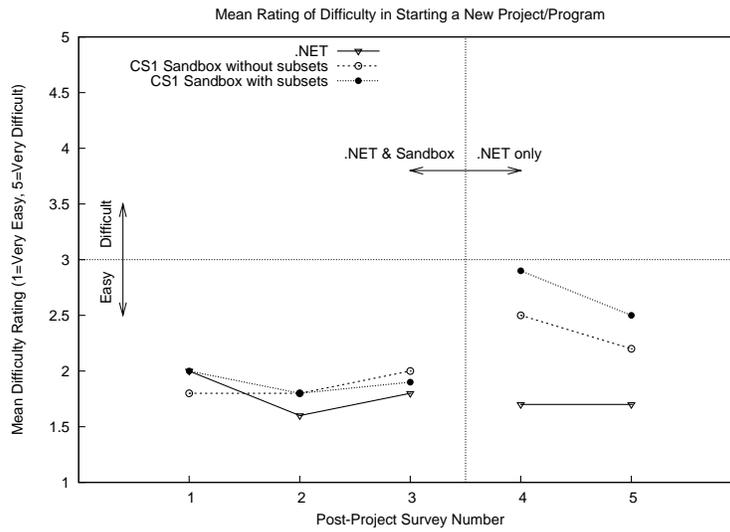


Figure 5.10: Graph of mean survey ratings of difficulty in starting a new project/program

Table 5.12: Mean rating values from the post-project survey question: *How would you categorize the difficulty in entering code and debugging syntax or semantic bugs in your project solution using this programming environment? (Very Easy=1, Easy, Medium, Difficult, Very Difficult=5)*

	Post-Project Survey				
	#1	#2	#3	#4	#5
.NET	2.2	2.1	2.1	2.2	2.1
CS1 Sandbox without subsets	2.7	2.7	2.9	2.4	2.1
CS1 Sandbox with subsets	2.5	2.5	2.7	2.3	2.3

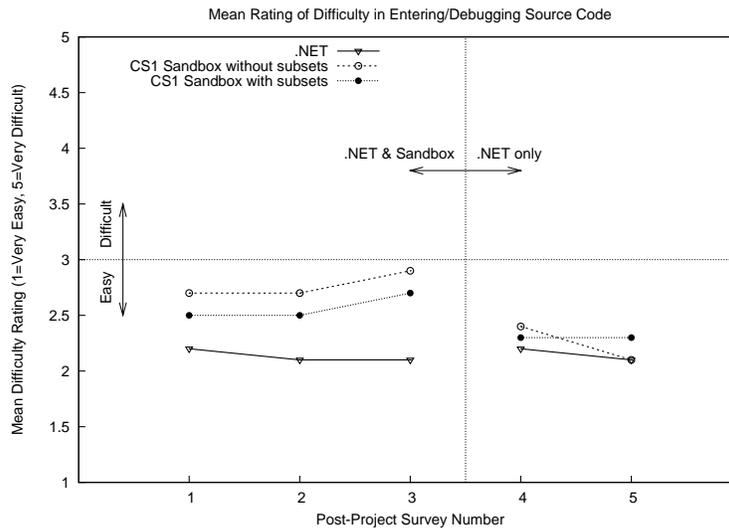


Figure 5.11: Graph of mean survey ratings of difficulty in entering/debugging source code

Table 5.13: Mean rating values from the post-project survey question: *How would you categorize the difficulty in testing your project solution using this programming environment?* (Very Easy=1, Easy, Medium, Difficult, Very Difficult=5)

	Post-Project Survey				
	#1	#2	#3	#4	#5
.NET	2.1	1.9	1.9	2.2	2.0
CS1 Sandbox without subsets	2.3	2.5	2.8	2.1	2.2
CS1 Sandbox with subsets	2.4	2.4	2.6	2.3	2.3

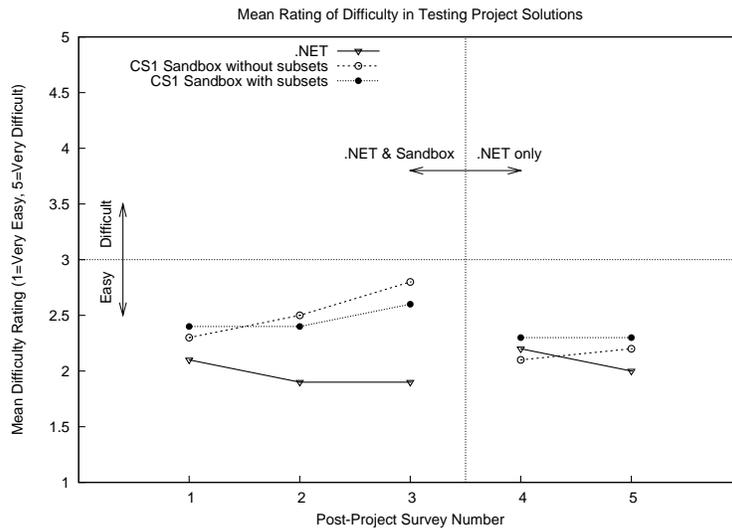


Figure 5.12: Graph of mean survey ratings of difficulty in testing project solution

Table 5.14: Mean rating values from the post-project survey question: *How would you categorize the difficulty of the project? (Very Easy=1, Easy, Medium, Difficult, Very Difficult=5)*

	Post-Project Survey				
	#1	#2	#3	#4	#5
.NET	2.1	2.5	2.5	3.6	3.0
CS1 Sandbox without subsets	2.3	2.3	2.5	3.5	3.0
CS1 Sandbox with subsets	2.4	2.2	2.4	3.4	3.1

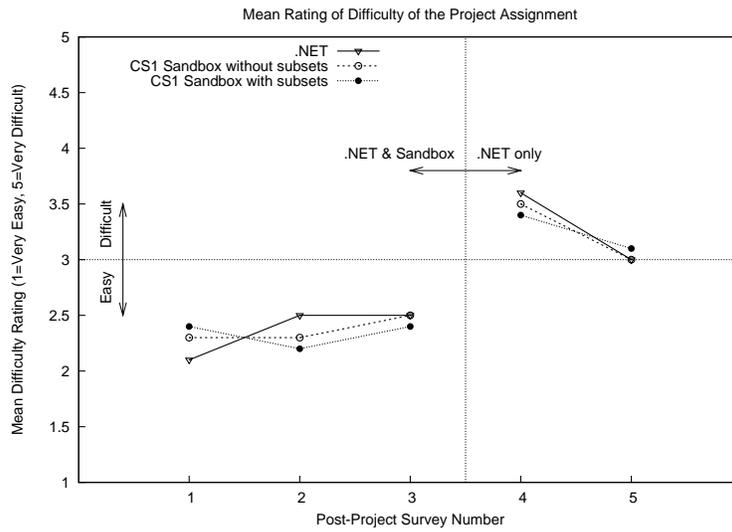


Figure 5.13: Graph of mean survey ratings of project difficulty

Table 5.15: Mean rating values from the post-project survey question: *How often did you use the in-environment help? (i.e. MSVC Help, Sandbox help) over the course of completing the project? (Not at all=1, 1-5 times (overall), 5-10 times (overall), 10-20 times (overall), All the time=5 (so many, I can't think to count!))*

	Post-Project Survey				
	#1	#2	#3	#4	#5
.NET	1.2	1.2	1.1	1.2	1.1
CS1 Sandbox without subsets	1.6	1.4	1.4	1.5	1.3
CS1 Sandbox with subsets	1.7	1.2	1.4	1.5	1.3

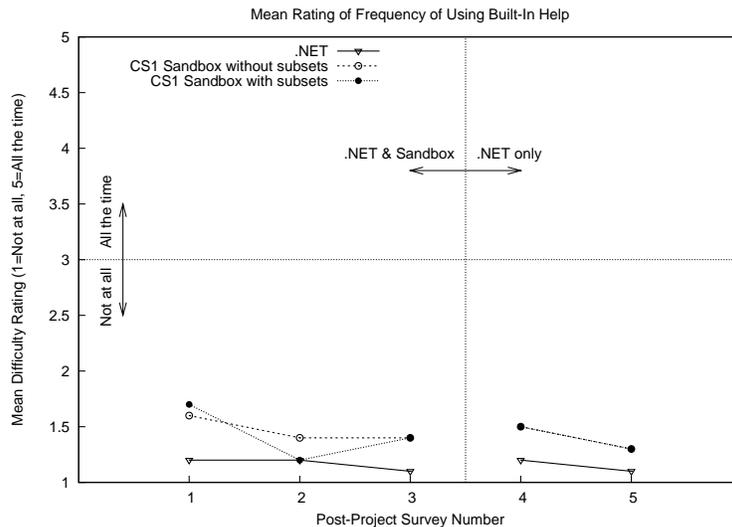


Figure 5.14: Graph of mean survey ratings of the frequency of use of built-in help

5.2.5.3 Write-in Data

In addition to the above mentioned rating survey questions, the post-project surveys also included two (optional) write-in questions as follows:

- “What features of the environment did you like the most, or were most helpful to you?”, and
- “What features of the environment did you dislike the most, or cause you the most confusion?”

These write-in questions were provided as part of the surveys to assist this researcher in obtaining the subjects’ thoughts related to the positive and negative features within their programming environment not already surveyed. All tolled, approximately 123 comments were provided about features or characteristics that the subjects liked, and 160 comments were provided regarding features or characteristics that the subjects disliked.

Table 5.16 provides a listing of the number of responses by project. It should be noted that responses from the surveys following projects #4 and #5 deal with the use of the .NET programming environment, as all subjects who has previously used the *CS1 Sandbox* programming environment had been migrated to the .NET programming environment for both of these projects.

A selection of subject responses appear in the tables that follow. Tables 5.17 through 5.19 are comprised of comments from the first post-project survey undertaken – obtaining the initial impressions of the programming environment by members of each experiment group. Comments from the second and third surveys were found to be very representative of the comments highlighted below. Table 5.20 follows these initial sets of comments with additional write-in remarks from subjects who used the *CS1 Sandbox* programming environment (both without and with subset support) following their migration to the .NET programming environment⁶.

⁶All spelling and grammatical errors from the write-in comments were preserved to reflect the recorded comment by the subjects.

Table 5.16: Frequency of responses to the write-in post-project survey questions

Post-Project Survey #	# of positive comments	# of negative comments
1	66	73
2	19	30
3	14	29
4	13	17
5	11	11
Total:	123	160

In all of the data presented from the write-in survey comments, this author has attempted to extract and present responses that span the various comments submitted during the surveys. In presenting this information an attempt was made to avoid listing comments of a similar nature regarding a similar preference (like or dislike) of a similar feature or characteristic (ease of use, easy to start a project, help messages present to users, etc.). Where duplicate comment exists for similar characteristics, they have generally been condensed to one representative comment and marked with a multiplier indicator (e.g. (x3) to indicate that two additional comments of a similar nature were made but remove for condensing purposes). It is believed that by condensing the comments in this fashion a wide cross section of responses can be presented.

Table 5.17: Selected Write-In Comments from the Post-Project Surveys - .NET Users

.NET	
Likes:	<p><i>"had no idea there was in-environment [help]; will definatly be using that next time"</i></p> <p><i>"I just liked how .NET was set up; it was very easy to use" (x2)</i></p> <p><i>"The bolding of braces and parentheses was helpful." (x4)</i></p> <p><i>"The debugger was great! I had a division by 0 error that kept crashing each time I executed and it found it right off and fixed it." (x4)</i></p> <p><i>"I like the layout of the different windows as tabs; and the buttons (however there are a mulititude of them)." (x2)</i></p> <p><i>"Double clicking on an error comment will jump you to that error in your code." (x3)</i></p> <p><i>"i think Visual.net is superior for writing and debugging programs compared to all others i have worked with. I like the Quickwatch feature so i can read my variable values in as im going through the code."</i></p> <p><i>"The automatic indenting that .net does is really cool and does not get in the way as much as autoformatting in other microsoft products." (x2)</i></p> <p><i>"I liked the ability to move the mouse and select a line without having to scroll to it; as well as decent copy/paste. It was also very convient to be able to write a few lines of code; save them; compile them; and test them all in very short order."</i></p> <p><i>"reserved words and documentation were color coded"</i></p> <p><i>"easy-access tools; automatic tabs sets; ability to change input file and view output file within the application"</i></p> <p><i>"The resemblance to Visual Basic made the environment seem familiar and didn't overwhelm me with an entirely new UI."</i></p>
Dislikes:	<p><i>"Not too many; really; just a few archaic error messages and a longer load time because .NET has so many things it needs. My biggest gripe has always been that VS will give you every possible error message and won't say "Error at this point; which caused the following errors;" you know; when 1 error causes about 20 others; and fixing the 1 fixes everything. It would be nice if it was a tad more intelligent."</i></p> <p><i>"I wish there was a build/execute button; (or that i knew of one) instead of clicking on two things. I also wish that it just jumped into a C++ environment every time i started a new program (in the labs) instead of having to select what type of app i want; and what language i want to use."</i></p> <p><i>"I hate with a passion the way that Visual Studio.NET starts up and the project creation process. It's too slow." (x3)</i></p> <p><i>"Creating a project and cpp in that project. If it is not done right(place the cpp out of the project); sometimes I will be unable to build the cpp by itself."</i></p> <p><i>"The errors statements are not as clear as they could be" (x4)</i></p> <p><i>"The screen seems to clotered with options"</i></p> <p><i>"Unclear error messages such as.. "... unexpected end of file found before end of file." I didn't know exactly what this meant and I aske the TA and they said that Visual.Net does that sometimes so I copied my source file and opened a new project and my program compiled just fine. I still don't know what caused the bug. Maybe the errors could be a little more specific on what line the failure actually occured and what caused it. " (x4)</i></p> <p><i>"eh... the help is like 1 Gig to install; so it would be nice to have a smaller to the point help file in .NET; but the course notes work fine."</i></p> <p><i>"The debugging procedure was complex and difficult to wade through; so I didn't use it. Other than that the environment worked out well enough."</i></p> <p><i>"The tooltip popups for functions like ignore-the parameter specifications are so complicated that I'd rather just go without them than have them in my way"</i></p> <p><i>"some of the pre-set indentations and allignments"</i></p>

Table 5.18: Selected Write-In Comments from the Post-Project Surveys - Sandbox Without Subsets Users

<i>CS1 Sandbox without subsets</i>	
Likes:	<p><i>"I liked the fact that the overall layout of the compiler is simple; basic; straight-forward. Unlike other compilers that are confusing with the extras and all; this is just pure coding. As a beginner; I feel that this is helping me much more than relying on the extras from other compilers. I am happy to actually learn the coding and see it simple and plain; because it will help me later with other programming."</i></p> <p><i>"I liked the simplicity of the environment; with other compilers there were too many buttons and drop down menus that it sometimes got confusing. With this environment I didn't have to worry about working with five different windows that showed my classes; header files; functions; things of that nature just get in the way and clutter things when I am just starting to program. I also liked the fact that the programs would compile right away; I wouldn't have to create a project first; which then created more files; before the program would run and show me the results/bugs." (x9)</i></p> <p><i>"I like how it displays an asterik next to the filename if any changes have not been saved."</i></p> <p><i>"The help file is pretty well done; the one time that i had to use it; my problem was easily fixed. I thought that made the enviornment rather easy to use; besides the fact that it is simplistic and straight forward; a nice deviation from the microsoft standard."</i></p> <p><i>"Nice easy startup. I've used .net before and I think its ridiculous. They should have left it like VS6. At least Sandbox cut out all the bells and whistles and just lets you make a source file easily."</i></p> <p><i>"The way everything was color coded to help you know if you typed it in correctly."</i></p> <p><i>"That we always had the latest version without having to download and install over and over. The large compile button b/c I always like to compile large chunks of code before I move on. The integrated console that docks instead of overlays."</i></p>
Dislikes:	<p><i>"at the same time... i could never really see exactly where my errors were in my programs"</i></p> <p><i>"Lack of scroll buton support; It's a must!" (x2)</i></p> <p><i>"There is no "undo" ability; and often times the color coding is not consistent."</i></p> <p><i>"The line coloring is buggy and needs to be worked on; because it's such a help when debugging code; on a high resolution monitor."</i></p> <p><i>"Sometimes when errors would pop up in the window underneath the code the column and line identification would be wrong; which confused me. I had to search through the code and find exactly where the problem was." (x2)</i></p> <p><i>"The bugs associated with line highlighting; ambiguous error messages; lack of support for one-line filestream initalization"</i></p> <p><i>"Not all help messages are documented. Compiler seems to have issues with certain call to <iomanip>. Environment is noticably slow at times (probably a Java thing; though.)"</i></p> <p><i>"I CANT RIGHT CLICK!!! The colors were messing up a lot I click on the error; it highlights the WRONG SECTION"</i></p> <p><i>"...sometimes the cursor would just disappear and the only thing that would get it back was a successful compile of the program which explains why I hit compile so many times. ...it took me almost a half hour of randomly commenting out lines to find out that sandbox did not support the abs function. It didn't give me an error when I used the abs function twice; but when I had one use commented out and only one remaining it did give me the error. I honestly think using VS6 or .net like I used in high school would've cut the project time in half."</i></p> <p><i>"On a newline; instead of it automaticly indenting to where the last line was it goes back to the beginning. I like it so it indents to where the last line was indented. Also sometimes the color scheme would go goofy." (x3)</i></p> <p><i>"The setprecision didn't work the same with sandbox as it did with the curator and caused some confusion which was only fixed after I read the forum."</i></p> <p><i>"The ignore statements are difficult to use. It gives the correct output but when the curator grades it the output obtained is undesirable."</i></p> <p><i>"Syntax highlighting was a bit weak. Adding bold facing would help. No automatic tabulation after another line that was tabulated. No find and replace feature. I like to output to the console to check the output before having to output to a file so that I dont have to keep opening and closing the file to refresh the output."</i></p>

Table 5.19: Selected Write-In Comments from the Post-Project Surveys - Sandbox With Subsets Users

CS1 Sandbox with subsets	
Likes:	<p><i>"I enjoyed the simplicity of Sandbox. While it may seem "dumbed down" to some; it is perfect for beginners such as myself." (x14)</i></p> <p><i>"Color coding correct syntax."</i></p> <p><i>"I like the big buttons; they are fun to press."</i></p> <p><i>"The environments inability to do stuff that is worthless to me."</i></p> <p><i>"there is a great deal of work put into sandbox and i respect that."</i></p> <p><i>"Very simple to use. Cuts a lot of the "professional" features of .NET which aren't needed at the academic level; which makes it much simpler to focus on the code."</i></p> <p><i>"- it's clean and simple - we automatically get the newest version"</i></p>
Dislikes:	<p><i>"The environment is great. It really helps the guy who's never programmed before. I would say its only downside is being slow. I suppose its an inherent flaw in JAVA but it seemed like backspacing or typing lagged behind the speed of my typing. I found myself typing code in a text file; and then just copying it to run it." (x2)</i></p> <p><i>"I think that I would add a print option to Sandbox; allowing the user to print out the code and look at it away from the computer. I would also add cut and paste options in the menu. This would be helpful to those who do not know the keyboard commands."</i></p> <p><i>"I found some of the error messages very vague; perhaps ambiguous. This made me cipher the information myself; scrutinizing the lines of code more thouroughly. Honestly; this made the programming experience better. I had to look harder at my code to see what was wrong. However; with more difficult projects; I certainly do not want to have to go through thousands of lines of code to figure out what an error message really means."</i></p> <p><i>"No auto tab; slow running; syntax highlighting is buggy; missing functionality; lack of control. * NOTE : I don't need auto tabulation after and ; just having something to allow me to tab blocks of text and have it remember the level of tab I am at would be great. *"</i></p> <p><i>"Small window was a little inconvenient because resising causes program output to not be seen."</i></p> <p><i>"I found the help file to be confusing in that it kept saying I had a parsar error. It could just be that I don't know what that is; but it isn't to helpful for newbies at programming."</i></p> <p><i>"The cursor tracker at the bottom of the screen does not match up with the line and column labels on the error messages so it is hard to find where exactly your errors are." (x3)</i></p> <p><i>"Bugs; few and far between; but every once in a while....."</i></p> <p><i>"In the sandbox there is no right click menu to copy and paste. In sandbox if you remove code it doesn't see it as editing so the 'save' remains dissabled."</i></p> <p><i>"- the mouse wheel doesn't work...hehe - can't create an exe; so it's cumbersome to test various input files; need to compile each time; instead of just running the exe"</i></p> <p><i>"Debugger vague."</i></p> <p><i>"just a few bugs that the compiler didn't catch and curator did. I wasted 3 submissions because it wouldn't compile on curator"</i></p> <p><i>"like all other porgrams the debugging needs work; hte col and line did not work most of the time. It needs to tell you what is happening when a program is running such as failed to output or out putted to screan ran sucessfully and things such as that. The program crashed several times and a few it took my computer with it. Some times you would get different output if you ran it 2 time whether it ran the first time or not I don't know if is just didn't run or what. Full implementatiuon would be nice for people who have programming experience"</i></p> <p><i>"the subset feature was confusing...but i understand that it was neccesary. Sometimes; however; I would do all my code and compile and i had forgotten about the subsets. Not a huge problem; but a little bit of an irritation"</i></p> <p><i>"subsets; language not fully implemented"</i></p> <p><i>"I don't like having to download subsets; and there was a small issue with printing zeros."</i></p> <p><i>"I dislike the restraints set by the environment. For example; when one needs to use coditional statements; or absolute value functions the most; they are not enabled."</i></p>

Table 5.20: Selected Write-In Comments from the Post-Project Surveys - Migrated Users

CS1 Sandbox (without and with subset support) – following migration to .NET	
Likes:	<p><i>“Auto indentation, auto completion of functions, the debugger, highlight of closed braces and brackets.”</i></p> <p><i>“When Writing Functions, It Has A Little Popup Box, That Comes Up Telling You The Parameters. It Also Indents Your Code For You.”</i></p> <p><i>“i didnt use the help much, but its awesome. i always enjoy the debugging.”</i></p> <p><i>“The debugger, although confusing at first, was very helpful.” (x2)</i></p> <p><i>“when using brackets, braces, or parenthesis, they bold for an instant to show you which ones match up, when calling a function, the prototype pops up to help you call it correctly”</i></p> <p><i>“all files in one window, descriptive error messages”</i></p> <p><i>“the good formating and the colors of the text.” (x2)</i></p> <p><i>“Was easy to run programs right out of dotnet.”</i></p> <p><i>“In this project I found the drop down menu of functions to be useful. When I selected a function it too me right to the function, which allowed me to easily move from one function to another when making changes.”</i></p> <p><i>“How it completes the end of certain phrases such as the variable inside a struct just by typing its first letter or scrolling down. A time saver.”</i></p> <p><i>“the classes view, i can just double click on one of my function in this toolbar, and the code window jumps to where it is”</i></p> <p><i>“ability to open all of the input and output files on the same screen as my cpp file.”</i></p>
Dislikes:	<p><i>“The over-abundance of features in .NET made it slightly confusing when switching from Sandbox. Also, it was hard to get use to debugging with .NET at first.”</i></p> <p><i>“Some of the error messages were very abstract and hard to understand, in particular the error messages I got when I had mismatched parameters between my function declaration and prototype function. Also .Net is more difficult to get a project started and it takes more time to actually compile and run the program than in the Sandbox environment.”</i></p> <p><i>“That little window that pops up everytime you start the program asking you if you want to change the output file because it has been modified outside of the environment. That, and the startup window is completely unnecessary.”</i></p> <p><i>“That I had to start a whole new project rather than just try to compile a single cpp file.”</i></p> <p><i>“It should be much easier to start a project. Too many buttons that are never used just clutters the sceen.”</i></p> <p><i>“For testing and debugging the auto-save is a bit of a pain. You can do undo but thats a little annoying. You should see if you can get undo in sandbox.”</i></p> <p><i>“I found it difficult to remember, after using Sandbox, that you had to compile and then run the program. It would trip me up a couple of times until I remembered.”</i></p> <p><i>“i don’t really know how to use the debug features...”</i></p> <p><i>“starting the program and the debugging process” (x3)</i></p> <p><i>“I started in VI, and compiled with G++. Unfortunately, Linux handles line returns differently, so my program didn’t work right until I moved into Dotnet. Then Dotnet started giving me an access violation, with my third input file. Even though it was a simple mistake, the Debugger actually complicated things by pointing me in the wrong direction. This was more user error than anything else, though.”</i></p> <p><i>“Some debugging messages I didn’t understand but I figured them out.”</i></p> <p><i>“The error messages are sometimes vague and hard to understand, sometimes it is easier to stare at my code for an hour or two “than to try to decipher the compiler’s error message.”</i></p> <p><i>“Still annoyed by all the clutter. There are so many buttons and menu options that I don’t use. It could be easier to start a project too.”</i></p> <p><i>“a few of the error messages”</i></p>

5.3 Error Recognition Results

Following the completion of use of the *CS1 Sandbox* programming environment, subjects previously using the *CS1 Sandbox* were migrated to the .NET programming environment. This migration milestone occurred following the completion of the 9th closed-laboratory session and the completion of the 3rd out-of-laboratory programming assignment. As part of the migration activities, a group of 29 voluntary subjects participated in short error recognition experiment.

The error recognition experiment involved the identification of syntax, semantic, and logic errors in the form of a short test (see Appendix G). All participants in the volunteer group took the test at the same time, and were permitted as much time as needed to complete the recognition. This author's notes indicate that the test took no more than 30 minutes for the last of the subjects to complete. Subjects who completed the test prior to their peers were allowed to leave the room or have a soda (the error recognition experiment was the prelude to the focus group's other activities).

The test presented a number of short programs and code fragments in which a subject would indicate the error (or lack thereof). In the test, there were a maximum of 14 potential errors spread among 13 programs/fragments. The errors were categorized and grouped as shown in Table 5.21. Table 5.22 shows the results of the error recognition experiment (the test) and categorized according to the experiment groupings.

Table 5.21: Error Experiment Classification Breakdown and Frequency.

Classification	Frequency of errors
Syntax	5
Semantic	4
Logic	2
None (no error(s) present)	3

Table 5.22: Error Experiment Data Collected – Number of Correct and Incorrect Responses by Experiment Group and Category of Problem

Error Classification	Experiment Group		
	.NET	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets
Syntax # Correct / Incorrect	47 / 8	28 / 7	46 / 9
Semantic # Correct / Incorrect	29 / 15	17 / 11	29 / 15
Logic # Correct / Incorrect	7 / 15	2 / 12	7 / 15
None present # Correct / Incorrect	22 / 11	14 / 7	22 / 33

5.4 Focus Group Transcript

A transcript of the focus group session⁷ (see Section 4.6.4) is provided below. Due to the large size of the room used for the focus group meeting, and the presence of rather noisy heating and air conditioning units in use at the time, several portions of the audio recording were inaudible. Since each subject was treated anonymously during the focus group, identified only by their programming environment, it was not possible to solicit individual subjects for clarifications of their comments.

Peter: *Let's start with what you liked about your particular programming environment.*

Respondent: *(Sandbox without subsets) I liked the fact that it was particularly streamlined around a certain area. I mean, I did a little bit of toying around in .NET, I didn't really like try to compile anything, but you know... it (.NET) struck me how complex it was to actually get your code to compile, compared to something else. You had to make a project, you had to do this, you had to do that. I guess because that's the point that you environment is specialized toward a specific group. I'm surprised .NET being used by those students, being you know, who had prior software engineering. The point is I like the fact it was simplified down the level where one can actually use it very quickly.*

Peter: *And you're talking about mostly the GUI – the user interface?*

Respondent: *Yeah.*

Peter: *OK. Others?*

Respondent: *(Sandbox with subsets) I agree with what he said. Having used Visual 6 (Visual C++ version 6.0) and .NET before coming here, and it was like a welcome change because it was all about the code. You didn't have to worry about debugging or anything like that...or getting a project up and running.*

Peter: *Like project management kind of things?*

Respondent: *Yeah, exactly. I mean it's fine if you are in like a professional environment. I think that Sandbox performs really well for a educational environment.*

Peter: *OK.*

Respondent: *The other thing I liked was the subsets. I thought it was a really neat idea.*

Peter: *Really!*

Respondent: *Yeah. Because I think it works well to provide like a nice structured way of how you're trying to use the language. That way it kind of forced them to get it right. For instance...I learned C++ in high school, but coming here I realized there were a bunch of things I had learned – there were more than one right way to do them. I think that if you can take the subsets even further could be really beneficial. (inaudible)*

Peter: *Just as a side note, you kind of hit on something – that's where the name of this thing came from. We wanted to build a pedagogical sandbox for you to basically play in. So that's where Sandbox came from. The other side of that – we want to be able to do – one of the further things I would do is extend the whole subset notion and be able let the instructor say “debugging is not 'on' until we turn it on”. Things like color syntax highlighting, we didn't get it done until the very first week of the semester, but it would have been cool to say “Yeah that's there, but we're not going to give it to you until you are comfortable programming... and then we're going to throw this other thing at you which may or may not confuse you”. So, we*

⁷The focus group session was held at the point where subjects who previously used the *CS1 Sandbox* programming environment were migrated to the .NET programming environment

can take that idea of turning things on and off and widen it into a much bigger arena as well. Other comments about stuff you all liked or didn't like?

Respondent: (Difficult accent and low voice volume, respondent is inaudible.)

Peter: So you are comparing it to command line compilation. So you like the complete package – the fact that you can work completely in one software application?

Respondent: Yes.

Peter: To do all the different things you need to do?

Respondent: Yes.

Peter: OK. Other thoughts, comments, things that were good?

Respondent: (Sandbox with subsets) I miss the search and replace function.

Peter: That was one of the things we didn't have time to do. Others (comments)?

Respondent: (Sandbox with subsets) I found that it was a lot easier. I didn't have to worry about all these different things like all this (inaudible) C++ Visual 6, as our programming environment. I found it pretty easy to focus on code, like (inaudible) That was our primary focus – we didn't have to worry about all the extra things (inaudible) all the unnecessary stuff.

Peter: Anybody else? Like from .NET?

Respondent: (.NET) ...like a lot of people I used Visual C++ before. One thing (inaudible) copy and paste (inaudible) color syntax highlighting (inaudible) that's cool.

Peter: That brings up a good point that I wanted to ask everybody... Did anybody not...let's assume you all liked these features, but did anybody...and some of them are in my environment and some are not... Things like color syntax highlighting. Is there anybody that did not like that? You know what I'm talking about, when I say color syntax highlighting? You know, you are typing and keywords are green, red, and then like string literals are a different color, numbers are another color, things are bold...

Respondent: (Sandbox with subsets) ...programming environment that just shut off and that got frustrating because (inaudible).

Peter: Yeah, for some reason it got lost and stopped (color syntax highlighting). One of the things I wanted to fix was that, but we didn't get time to do it. That's what happens when you use someone else's code and plug it into your system. Other things related to color syntax highlighting or in the case of .NET, and I know this isn't in Sandbox, but autoinserting...where it inserts the tab for you, or auto-indentation. Or brace checking? Or parenthesis checking? And stuff like that. Anybody not like that kind of stuff?

Respondent: Well I (inaudible) sometimes it would do that and sometimes you would press enter and and it wouldn't do that and you'd have to do it yourself.

Peter: You're talking about .NET, right?

Respondent: Yeah.

Peter: Good, because mine (environment) didn't do that. (Laughs)

Respondent: Sometimes I'd press enter and it not auto-indent I'd have to do it myself. I'd rather it always work or not do it and I'd do it all myself.

Peter: We tried to do an auto-indent thing, and there's a student who is interning at IBM and kind of working part time for me. He tells me it's really hard and I haven't looked at the problem on how to calculate and manage that stuff. But I'm going to take his word for it because he's a pretty good coder. So, it's actually a hard problem to deal with I think because

there has to be some intelligence built into it. And the other part of it that's annoying, especially in my environment in Java, is you have to do it quickly. You can't just take 30 seconds to figure out what color something should be or tabbing, because the users go: "I can't deal with this, I'll just do it another way."

Respondent: (.NET) ...I thought that the auto-indenters were consistent and it was helpful usually, because if it didn't indent when I hit enter then it then it should not have done that for (inaudible) some parts that I didn't need to be in. Other than that, I thought it really (inaudible).

Peter: For the people that used my environment, would a feature that auto-formats your code be desired or undesirable? Meaning you typed in a bunch of stuff and you go to a menu item that says "Format all this stuff for me", and it does all the lining up and tabbing and stuff like that... Yes?

Respondent: (Sandbox with subsets) I feel that, I mean, I think that's sort of against the format of what you are trying to do. I mean if you are trying to create a coding environment for students part of that is learning programming style. And you said it yourself, you didn't have much time to do this before they were pushing you along. Basically, auto-indenting... sure it would be nice, but you know the thing is, one, it can screw up - even Microsoft screwed it up sometimes and, two, it would take a lot of implementation, but, three, more to the point, I mean people should learn how to indent their own code.

Peter: Yeah but ask these two (teaching assistants) about how much people lost or are losing (points on project) because they are not following what we've advocated. I know what you are saying because it's a learn-by-doing experience, but there are some people that just, I have found, have a conceptual problem with understanding what we are talking about in terms of making that happen. Other thoughts?

Respondent: On the auto-format thing, personally, I don't really like it, only because people you know, pick different styles when they are coding and like in the notes you said before, I know that usually Mr. McQuain opens all the functions with the brace on the same line and I've always done it on the next line, and stuff like that. I guess it's kind of nit-picky stuff, but I just think that, like, the auto-format to me seems like a gimmick instead of a feature. I know that if you look at .NET... I've never used it but...

Peter: Yeah, I've heard that .NET actually has this now that somewhere you can pull down a menu and it does it, but I haven't played with it.

...

Peter: Alright, let's switch to a different topic. What did you not like in your respective programming environment?

Respondent: This is a small one. I used .NET and the little start window that it brings up whenever you open it...

Peter: You mean the start up window? (Laughs from group)

Respondent: I feel that when you get to the level where you are writing any kind of program at all, you should be able to get past like, all the little (inaudible) to the last thing that I worked on. So far (inaudible)...

...

Peter: Anything else that you didn't like?

Respondent: (Sandbox without subsets) Um, the error messages... weren't very helpful.

Peter: Gotcha. Which ones?

Respondent: I mean, (another respondent comments)... yeah, "expected this found that". You know it would be helpful if it said you missed a semi-colon here. That might be more helpful for the people that are just starting out with this (programming). You know, going back to who this is intended for. I would get it after a second, but some people they wouldn't. And so, I mean, saying "you need to add a semi-colon here" might be more helpful than "expected this, found that". I know it's really hard to write error messages, and stuff like that. I mean, the error thing that you have is really good, it actually highlighted the line, and everything. Compared to what other things have.

Peter: So you'd like to see more detail with the error message, and less ambiguity.

Respondent: Yeah, less ambiguity.

Respondent: One of the things that I noticed about specific error messages. I'm not sure if it was (inaudible)...it just seemed kind of odd to me, when I tried to use functions and it didn't have the right header (ed: function prototype) included it would tell me "cannot locate function in current scope". And that confused me because it gave me the impression that somehow the function is there, but not in the right scope. It took me a while to realize that (inaudible) that specific error (inaudible) The only other problem I had, I think that (inaudible) the line column thing didn't seem to be working.

Peter: Yeah, throughout the semester we had numerous issues, and most of it goes back to the end of line in Windows and the end of line character in Unix and Macs are all different. So, to be able to... we tested everything under Unix, more than we tested under Windows. And when you folks got it, we had all kinds of wacky problems with "that's not the right sequence of where the end of the line is." So it took us a while to sort of shake that out and get it right.

Respondent: (Sandbox with subsets) The only real problem I had was ... for the last project, my program compiled and everything worked perfectly in the Sandbox, but when I uploaded it to the Curator and it compiled it with .NET I had tons of problems. I think there are some errors in functions...I don't think you fully implemented functions... (inaudible)

Peter: There were some changes in the last two weeks with functions. Some things were broken, and they didn't get fixed until it was almost too late. So in hindsight, we probably should have not tried to make the changes we did, but that last week was kind of rough for some people because there were things that were supposed to get caught and didn't.

Respondent: (inaudible)

Peter: So that you are doing some expressions, putting in variables, combining variables, so that finally instead of making one big long thing. That I would be interested in seeing. Do you happen to know if it had to do with negation?

Respondent: I think so, I printed it out.

Peter: Yeah, I wouldn't mind taking a look at it.

Respondent: That happened too when I (inaudible)...

Peter: OK, those I'd be interested in seeing.

Respondent: (inaudible) ...

Peter: Really! OK. Yup?

Respondent: (Sandbox without subsets) I just thought of something. It was just a slight annoyance. When it's saving a file...I guess this is my peculiarity, I got used to how Microsoft auto-extends file names, it will automatically suffix it with an extension. So you know when you see the save box there, and you type answer.cpp or whatever - I type in my project name and hit enter and...

Peter: That's the name you get...

Respondent: Yeah, that's the name you get.

Peter: Right.

Respondent: I mean, again, it's probably just me, but you know, somebody else...at least some other person would see that and blow up.

Peter: OK.

Respondent: One of the things that I noticed about saving is that like, the whole thing where you change the file and you are allowed to save it. I think it would be helpful if you could save even if you hadn't changed it. Just cause, I don't know, like I always save before run a program (inaudible and audience laughing).

Peter: OK, but how does saving it when nothings been changed - how does that solve that?

Respondent: I don't know I could be wrong, but noticed that some times I would type some stuff where actually, maybe where I select stuff and hit paste...

Peter: OK, now there was a situation where if you paste or possibly cut... Cut, copy, and paste - the functionality is there in that it's part of the typing area, but there's nothing found between the actual action and me telling you that the file has been modified. So, again, it's one of those things that was on the list but we never got to it. Actually, I had this situation where I pasted something in and it looked like the file hadn't changed, so you had to actually go in and type a few space characters, and then delete them, and then save the file. It's a quirky little thing. Yes.

Respondent: (inaudible)

Peter: OK.

Respondent: (inaudible)

Peter: Yes?

Respondent: I used .NET. I had problems with function prototypes didn't match the definition. The error I got was an 'external error'. I just thought it could have been put into a better form. It was just a matter of the type of one of the variables being off.

Peter: That's a linker message. This starts to show a difference between my environment and their environment. We do complete interpretation and they do compilation, and they compile the file or files - and you'll get to the point where you'll be doing multiple C files, and then there's another pass where the linker comes in and links all - you are calling this function and that's in this library and those are in external files. Like if you are calling the sine function and you spell it wrong, it won't know until that phase that there is something wrong. The problem is that the message for that is God-awful it's 'extern underscore underscore dollar at at at sine' and you go 'Whoa, hey!' The first time I saw that and I was helping people I was like 'What the heck is this?' Now when you see it it you are like 'OK, I know what that is...it's a problem with the ya' know'.

Respondent: It was hard because it didn't even give me a line (number) or anything, I've been playing with the code, but wasn't sure if it was the prototype or the definition.

Peter: Right. OK, others?

Respondent: (Sandbox with subsets) I didn't think it was very helpful to have a console in there when everything we were doing was with files and there was no console stuff at all. Another thing I thought would have been useful would had an interpreter log so that you know what it's doing...

Peter: That's in there (a debugging window), but I didn't tell everybody about it. (Laughs from other subjects). No, well, see the thing is that there's a whole window you can pop-up. You hit Control-2 and the window pops up and you get three different windows for debugging. One is for the semantic pass, one is for the interpreter pass, and the other is just standard output junk. The problem is that it's pretty complex and for the most part it may or may not tell you guys anything useful and thought it was a better idea just to hide that - but I needed it so that when you came to me and said that this is broke, at least I had something to look at other than 'Unknown error'. Which, you know, tells me nothing as well. Um, so there, yeah, I can see, I know what you are saying, but it might be too dangerous to give it to you. Yes?

Respondent: (inaudible) a watch on a variable (inaudible)

Peter: Well no, that's for debugging. A watch on a variable is for debugging. What he was talking about is more like - you have a log where it's saying, 'OK, I'm checking this function, here's where I found a problem, and we can look and see 'OK, at this point right here there was something bad happening...'

Respondent: I used something called (inaudible) it put locks on things.

Peter: Yeah, that's really using the debugger and that's something I'd like to add in, but that's significantly more work to be able to track that, and a good debugger will let you get to a point and possibly back up which is really really tricky, but I know what you are saying. That's helpful. We don't teach you a whole lot about debugging in this course. You get a whole introduction to it - it's in the notepack, you get a little more work with it later on...

Respondent: (inaudible)

Peter: OK, so multiple text windows, or it being knowledgeable that you wrote to a file and then it gives you the option to show you the file.

Respondent: I used .NET. I like being able to maximize the window (inaudible) put the file on your screen (inaudible)

Peter: Yes?

Respondent: I noticed that if you maximized the input/output window it would block the code window or end up in the toolbar.

Peter: Yeah, that's a bug we tried to fix, and there's a really complex reason why. There's a problem we run into getting rid of those windows - and then it comes back up and it's always in the toolbar, so it's minimized, and there's some bad behavior with maximizing as well. I tried to remind people in the labs: 'Don't make those windows full-sized - you can stretch them a little, move them around a little', but probably needs some good debugging. Yes?

Respondent: (.NET) ...renamed my .cpp file and then in .NET the compile wouldn't (inaudible).

Peter: Yes, what you've done is when you make a file in .NET, you are putting it in a project hierarchy. OK? And you are telling it that this project has these files in it. Now, if you go outside the programming environment and change the filename, the environment is still looking for this file. So that sounds like what's happening there that in a sense the project management stuff is sort of defeating you in terms of simple management stuff.

Respondent: Yeah.

Peter: OK. There was one other... yeah?

Respondent: (.NET) ...my input and output files when I used .NET, I'd get the message that they had been changed by something other than .NET. (inaudible)

Peter: Oh, the output file changed do you want to reload it? Yeah. (Discussion among subjects). The only way you can fix all that is to not display the output window. What it's doing is as long as it's open and there are changes to it, its asking if you want to reload it. And that can be a big pain. If it's not displayed, but it's part of the project, when you want it, pop it up, look at it, kill it (the window), finish the debugging, pop it up again, etc.

Respondent: (inaudible)

Peter: I agree. I would be nice if you could just say "Hey, the thing is there, when it changes, update it and stop annoying me."

Respondent: (inaudible comments)

Peter: OK. Yes?

Respondent: (Sandbox with subsets) One thing I would have liked to have was a scrollbar at the bottom (inaudible)

Peter: Wait a minute, say that again. A scrollbar at the bottom?

Respondent: (inaudible) when you get to the end of the line and it goes to the next line (inaudible)

Peter: Oh OK, so there was no horizontal scrollbar so it did a wrap. OK, I must have missed that one. Yes?

Respondent: (.NET) When you build and get warnings - I wish you could tell it not to put it in the list any more. Because like, you might only one, but at first glance it looks like something is really wrong it makes you stop. And like when you are changing one thing at a time, and having to go back and do that over and over again (inaudible).

Peter: Right. OK.

Respondent: (Comments from other subjects)

Peter: I also wanted to know, has anyone used on-line help in either of the environments? Personally, I went through all kinds of pain to write 50 or 60 help files with examples, so I'm curious to know if anyone actually used those. But also similarly in Microsoft (.NET) where you can click on the error message and somehow, somewhere, I forget how it is with .NET, but you can get a help file that talks about a problem situation, a keyword, whatever the message meant. So let's talk about help for a minute and then I think we can wrap up. Does anyone have any thoughts on help?

Respondent: (.NET) We would try and go look for a specific thing like, for example, a while back I was trying to look up binary file i/o...

Peter: Yeah, we do a lot of that in this course (chuckles from other subjects)

Respondent: ...well not this course, but last year. They have everything there, but they don't have good examples at times, and then like the syntax, they output the commands and stuff. It isn't as helpful as it could be. But if you look at it the other way, they have so much stuff to write that it would be hard to do that stuff. It's not really geared toward the beginner, it's geared toward the people that already know a bit about the language.

Peter: Absolutely.

Respondent: So, the help files in .NET help, but they are not as helpful as they could be for beginners.

Peter: OK.

Respondent: (Sandbox) ...I used .NET prior, so one thing I noticed is that they kind of throw everything in .NET into the index thing, so if you are typing a specific function to lookup like the .NET framework or Visual Basic or C#, you might just hit enter by accident...

Peter: So you are saying you don't like how they index all of those things?

Respondent: I don't like how they indexed it all together. I think it should be indexed with the environment you are working with... (inaudible).

Respondent: OK. Yes?

Respondent: (Sandbox with subsets) One thing that I had problems with was that you only had certain functions implemented, so you were forcing us to use certain functions while others were not usable.

Peter: We definitely limited, especially in terms of functions that came out of header files or include files, to what we traditionally teach in this course and what we anticipated you folks using. I think the only exception to that was that somewhere we got down the road and some of the absolute functions were done, and some were not – so we had to whip those up real quick. In the beginning of the semester he (William McQuain) showed a slide about string concatenation, string length, and we don't do strings until the very last section of the course so I didn't do any of that stuff, and yet I found out later that "Oh yeah, there's this slide" and that six people had actually tried to go do that and of course it didn't work because we didn't know what the string length function was – so I whipped that up real quick. So, a lot of the decisions of what to do, and how to implement it was driven around what are you going to experience in this class and of course, if you programmed before, you are going to try to break things (the software) - I saw people doing all kinds of things such as C++ things and I was like "No, no, no! It's not going to work!" You know, arrays don't work. Somebody tried to do arrays last week, to start working on the next project which is good, it's just that not everything is going to work correctly at this stage. OK, you two folks, you both used Sandbox in one form or the other and you've switched now to .NET for the great changeover. What is your impression?

Respondent: Well, I guess (inaudible) didn't really have any problems with it (inaudible)

Peter: How about the number of menu items, panels, panes, subwindows, buttons, all that GUI stuff. Is that intimidating, or you're not so intimidated because you used Microsoft products before? (Laughs from students)

Respondent: I guess it could be a little intimidating (inaudible)

Peter: But you are getting through it? OK. How about your thoughts (other student)?

Respondent: I just like .NET (inaudible) the main thought was just on how many buttons there were.

Peter: OK. Other thoughts on anything else? That's pretty much what I wanted to extract from all of you – and get some thoughts on the different environments. Yeah?

Respondent: I think that .NET is different because it's hard to know everything that you can do. Like, file loader, 'cause right now (inaudible) If I just had one button that did everything and that was about it, I don't know if I could handle it.

Peter: Well, the rest of them seem relatively well adjusted. (Laughs from students) Yeah, but my impression is that you've had some prior programming experience, and you are probably not at the same level of experience that a lot of other students are in this class. In a sense, it's good that you ended up in the .NET group because you probably would have wanted to be there anyway.

Respondent: (Sandbox) I disagree though, (inaudible) would have been easier to figure out what you did wrong.

Peter: Any what did you use? .NET?

Respondent: I used Sandbox.

Peter: OK. Any other closing thoughts? OK.

Chapter 6

Analysis of the Collected Data

6.1 Statistical Methods Used in Analysis

Four basic statistical tests were used during the analysis of the data sets collected by this experiment. The selection and application of these tests were determined in consultation with the Virginia Tech Statistical Consulting Center during the semester the experiment was on-going, and semesters subsequent to the experiment. Each test is briefly described in the sections that follow, and includes a summary of the key test statistics calculated by each. Finally, each section identifies the data set(s) that were analyzed using that test.

For all of the statistical tests used in this analysis, the 0.05 significance level was used for rejecting the null hypothesis (means are equal).

“Traditionally, experimenters have used either the .05 level (sometimes called the 5% level) or the .01 level (1% level), although the choice of levels is largely subjective. The lower the significance level, the more the data must diverge from the null hypothesis to be significant. Therefore, the .01 level is more conservative than the .05 level.” [42]

6.1.1 Chi-Squared

Chi-squared tests for independence are utilized when the data under analysis is comprised of two or more nominal (categorical) variables. The data set used in chi-squared tests are frequency measures (counts) of the occurrences of each of the categorical variables for each experiment group under test. The chi-squared test for independence determines if the frequency measured for each variable and experiment group are contingent on each other.

A chi-squared test produces a chi-squared value (a value based on the observed (measured) frequencies compared to the expected frequencies), and a number of degrees of freedom (product of one less than the number of experiment groups and one less than the number of categorical variables).

Given these two values, a chi-squared table can be used to determine a corresponding probability value (P-value). If this P-value is less than the significance level selected (0.05 for this experiment), the null hypothesis can be rejected and the corresponding research hypothesis can be accepted (the frequency of the categorical variable is contingent on the experiment group).

The chi-squared test was used in the analysis in support of this dissertation as follows:

- comparison of the categorical demographic data (see Section 6.2.1) collected to ensure the experiment groups were equal (balanced) with respect to each demographic category (current class standing, gender, if the subject had just exited high school, and age), and
- comparison of the results of the error recognition experiment (see Section 6.5).

6.1.2 Analysis of Variance (ANOVA) & Duncan's Procedure

The analysis of variance test is used to test variations among the (numerical) means of several groups (two or more groups). More formally,

“Analysis of variance, frequently abbreviated as anova, is used in a large number of statistical procedures, including more complex experimental designs and prediction situations. In each application, the overall variability in the data is partitioned into components reflecting the various influences in the experiment that may lead to differences among observations. Tests are conducted to see if these influences have a substantial effect on the data.” [71]

“Hypothesis testing in analysis of variance is about whether the means of the samples differ more than you would expect if the null hypothesis were true. This question about means is answered, surprisingly, by analyzing variances”. [5]

ANOVA tests produce a number of interesting values in the process of computing a result. Among these are the degrees of freedom (both between and within the experiment groups), as well as an F-ratio (the ratio between the between-group and within-group variances) and a P-value (probability of obtaining a mean different from the mean specified in null hypothesis for the test). As with the chi-squared test above, the null hypothesis states that the means of the groups are equal, while the research hypothesis states the contrary.

In this experiment, the ANOVA statistical test is used to examine several categories of demographic numerical means. These include the following:

- comparison the mean SAT scores collected as part of the demographic data (see Section 6.2.2) used to ensure the experiment groups were equal (balanced) with respect to each demographic category,
- comparison of the mean closed-laboratory results (raw scores, number of compilation attempts, and number of errors obtained; see Section 6.3), and
- comparison of the mean out-of-laboratory project raw scores (see Section 6.4.1).

When a data set's statistical results indicate that the means of the groups are different, and when three experiment groups are involved in the test, Duncan's procedure was performed to indicate which of the experiment group means are different to the others. These results are indicated in the presentation of the results, where appropriate.

6.1.3 Kruskal-Wallis

The Kruskal-Wallis statistical test is the nonparametric alternative to the ANOVA test. Generally speaking, this test is used when the data is not normally distributed or the data set is comprised of nominal variables. Kruskal-Wallis test also produces a probability value (P-value) used to accept or reject the null hypothesis – as it is in the ANOVA statistical test mentioned above. Kruskal-Wallis tests were used in the analysis of the the five post-project survey questions where students ranked various experiences with their programming environment (see Section 6.4.5).

6.1.4 Repeated Measures

The repeated measures test is very similar to the ANOVA test described above. However, the design of the test attempts to account for the within-subjects variability (for example, accounting for a subject that tends to consistently perform a large number of compilation attempts). In doing so, the difference between the subjects is highlighted and measured. In performing a repeated measures test, all of a subject's results are tested together as one block of results (multiple measures). As with the ANOVA statistical tests, the repeated measures tests produce both an F-ratio and a P-value which are used to accept or reject the null hypothesis.

Repeated measures tests were used as follows during the analysis in this dissertation:

- comparison of the mean number of out-of-laboratory programming project compilation attempts (see Section 6.12),
- comparison of the mean number of out-of-laboratory programming project errors obtained (see Section 6.13), and
- comparison of the out-of-laboratory programming projects mean time-on-task values (see Section 6.14).

6.2 Statistical Tests for Parity of the Distribution of Experiment Subjects

To ensure the validity of this study, it was necessary to assure that the three experiment subject groups (.NET, *CS1 Sandbox* without subsets, and *CS1 Sandbox* with subsets) were equivalent with respect to their demographic composition. Such an examination seeks to ensure that no variable has more effect on one group than it does on another. For example, it is highly desirable to ensure that there are comparable amounts of female subject members in each experiment group. If the groups are comparable in such a regard, it can be said that no single group's results were affected by gender.

Comparison of the groups was undertaken for each of the variables captured in the demographic data survey in this experiment among the participants (see Section 4.4). The categorical data variables (gender, age, class standing, and high school status) were analyzed using chi-squared tests, while numeric variables (such as the Scholastic Aptitude Test (SAT) scores [12], American College Test (ACT) scores [1], and the programming experience scores) were analyzed using ANOVA tests.

Because of the low yield of subjects (17 of the 165 participating subjects) that had taken the ACT test in high school and due to the high percentage of those subjects who had reported SAT scores (149 of the 165 participating subjects), the ACT survey responses were not used in the comparison of the three experiment groups.

The results of all of the statistical tests pertaining to demographics executed are discussed in the following two sections. Conclusions drawn from these tests are discussed in Section 6.2.3.

6.2.1 Analysis of Experiment Group Parity - Categorical

In order to rule out potential effects as the result of an imbalance among the experiment groups, the three experiment groups were tested to ensure their independence with respect to the categorical demographic data described previously (see Section 4.4).

For each of these demographic categories a chi-squared test of independence was executed to validate that no relation between the categorical demographic variables (gender, age, etc.) and the experiment groups existed. Thus, for each analysis both a null hypothesis (\mathcal{H}_0) and a research hypothesis (\mathcal{H}_1) were formulated and tested using the chi-squared statistical test.

For example, in the first analysis executed below (see Table 6.1), the hypotheses are as follows:

\mathcal{H}_0 : There is no difference, on the average, in the mean class standing of the experiment's subject groupings.

\mathcal{H}_1 : The mean class standing of the experiment's subject groupings are different.

The resulting chi-squared statistic is $\chi^2 = 0.54$, and there are two degrees of freedom. For this analysis, the computed P-value is 0.76, well above the 0.05 significance level used to reject the null hypothesis. Thus, the null hypothesis in this case is accepted – there is no relation between the experiment's groupings and the mean class standing of the groups' subjects.

Table 6.1: Chi-Squared Test of Subject Groups vs. Class Standing (question #1 on the demographic survey form)

<i>Standing</i>	Sandbox with	Sandbox without	.NET
Freshmen	38	41	74
Sophomores	2	1	4

Chi-squared result: $\chi^2 = 0.54$

Degrees of freedom: 2

P-value: 0.76

Table 6.2: Chi-Squared Test of Subject Groups vs. Gender (question #2 on the demographic survey form)

<i>Gender</i>	Sandbox with	Sandbox without	.NET
Male	40	41	71
Female	3	3	7

Chi-squared result: $\chi^2 = 0.25$

Degrees of freedom: 2

P-value: 0.88

Each of the four chi-squared tests performed (see Tables 6.1 through 6.4) resulted in P-values greater than 0.05.

It is concluded that none of these categorical variables (class standing, gender, high school status, and age) differ significantly across the three experiment groups (.NET, *CS1 Sandbox* without subsets, and *CS1 Sandbox* with subsets). (Note that the fourth test (subject groups vs. age) results in a higher number of degrees of freedom. This is due to the additional dependent variable in this question.)

6.2.2 Analysis of Experiment Group Parity - Quantitative

In order to rule out potential effects as the result of an imbalance between the experiment groups, the three experiment groups were also analyzed to ensure their independence with respect to the quantitative

Table 6.3: Chi-Squared Test of Subject Groups vs. Exiting High School (question #4 on the demographic survey form)

High School	Sandbox with	Sandbox without	.NET
Yes	41	41	70
No	2	2	8

Chi-squared result: $\chi^2 = 1.95$
 Degrees of freedom: 2
 P-value: 0.38

Table 6.4: Chi-Squared Test of Subject Groups vs. Age (question #3 on the demographic survey form)

Age	Sandbox with	Sandbox without	.NET
17	3	8	10
18	32	30	52
19	5	6	10

Chi-squared result: $\chi^2 = 2.25$
 Degrees of freedom: 4
 P-value: 0.69

demographic information mentioned previously (SAT scores and prior programming experience; see Section 4.4).

For each of these demographic categories an ANOVA test was performed to validate that no relation between the quantitative variables and the experiment groups existed. That is, that the means of these demographic quantities among the groups were statistically equivalent. Thus, for each analysis both a null hypothesis (\mathcal{H}_0) and a research hypothesis (\mathcal{H}_1) were formulated and tested using the ANOVA statistical test.

For example, in the first test executed below (see Table 6.5), the hypotheses are as follows:

\mathcal{H}_0 : There is no difference, on the average, in the mean SAT scores of the experiment's subject groupings.

\mathcal{H}_1 : The mean SAT scores of the experiment's subject groupings are different.

An ANOVA test was performed on the SAT scores collected from the experiment group population to determine if the mean SAT scores are statistically equivalent among the three experiment groups. The resulting key values from this analysis (see Table 6.5) are as follows: the F-ratio is 0.83, the P-value is 0.44 and there are two degrees of freedom.

The computed P-value is 0.44, well above the 0.05 significance level used to reject the null hypothesis. Thus, the null hypothesis in this test is accepted – there is no difference between the experiment's groupings and the mean SAT scores.

The ANOVA test was also performed on the programming experience scores collected¹ from the

¹The self-reported programming experience scores were summed across each language for each student. This created an aggregate programming experience score for each subject.

experiment population to determine if the mean experience scores were statistically equivalent among the three experiment groups.

The resulting F-ratio and P-value from the test were 0.51 and 0.60 respectively (see Table 6.5). Since the P-value is greater than 0.05, the null hypothesis (there is no difference, on the average, in the mean programming experience scores of the experiment's subject groupings) was accepted.

Table 6.5: Analysis of Variance (ANOVA) Tests of Numeric Demographic Results (SAT scores and prior programming experience scores)

Data set tested	F-ratio	P-value
SAT scores	0.83	0.44
Prior Programming Experience	0.51	0.60

6.2.3 Demographic Conclusions

Based on the previously mentioned six statistical evaluations, it can be concluded that the experimental groups are equivalent in all of the respects captured by the demographic survey (see Table 6.6). This is a positive result for this experiment, as it is highly desirable to have each of the experiment groups be comprised of similar subjects (as far as possible).

Table 6.6: Summary of significant findings from the statistical tests on demographic data.

Statistical Test	Group means equivalent?
Categorical (Nominal) Tests	YES
Class Standing	YES
Gender	YES
Exiting High School	YES
Age	YES
Numeric Tests	YES
Mean SAT Scores	YES
Mean Programming Scores	YES

Given the nature of the distribution of subjects (subjects were permitted to enroll in the laboratory section of their choosing), there was no guarantee that this random distribution of subjects would not result in an unequal distribution of subjects (based on some demographic criteria).

For example, there was nothing to prevent all of the female members of the experiment (course) from registering for the same laboratory section. Though unlikely given the department's enrollment policy (self-enrollment during the summer orientation sessions), such a registration policy is occasionally undertaken by other computer science departments to combat the increasing attrition of female computer science majors by fostering a sense of community among the female students.

This is a positive result for the purposes of further comparisons among these experiment groups. By having groups which are statistically equivalent in nature, this result provides the ability to rule out interaction of any of the tested demographic categories in further tests. In other words, it cannot be argued that a statistically significant result (higher project scores when using the *CS1 Sandbox* with subsets, for

example) was the result of a particular experiment group containing a disproportionate number of subjects of a given demographic category.

6.3 Statistical Evaluation of Closed-Laboratory Results

As mentioned in Section 4.6, the closed-laboratory portion of the experiment resulted in the production of several data sets this dissertation needed to examine. These sets included the following:

1. the mean scores from assignments,
2. the number of compilations by each subject each week (laboratory session), and
3. the number of errors generated by each subject each week (laboratory session).

Of these three data sets, only the first (mean scores) could include those subjects using the .NET programming environment (recall that the .NET programming environment had no ability to elicit data collection results such as source code, errors and the like at the point of compilation/execution). The last two items (number of compilations and the number of errors generated) were obtainable only when subjects used the *CS1 Sandbox* programming environment.

6.3.1 Analysis of Closed-Laboratory Scores

ANOVA tests were performed on each of the mean laboratory scores from the first nine closed-laboratories, representing those laboratories which required the subjects to use their experiment-assigned programming environment. The raw mean scores (re-graphed in Figure 5.1 and previously presented in Table 5.1) spanned all three experiment groups (.NET, Sandbox without subsets, and Sandbox with subsets) and produced the ANOVA P-values shown in Table 6.7.

This test seeks to determine if there are unequal means among the groups with respect to the resulting scores obtained in the laboratory sessions. Numeric scores are a widely used mechanism for determining a student's comprehension of the material presented.

It is hypothesized by this author that subjects using a simplified interface (*CS1 Sandbox* without subsets) have statistically significant higher mean academic scores than their counterparts using an environment with a more complex interface (.NET). Additionally, it is hypothesized that subjects who have language subsets applied to their programming environment (*CS1 Sandbox* without subsets) have a statistically significant higher mean academic scores than their counterparts not using subsets (*CS1 Sandbox* without subsets). By executing an ANOVA analysis on this data, we can evaluate if any significant results have been achieved.

Of these closed-laboratory averages, only laboratories #1 and #6 (see the values in Table 6.7) contained statistically significant differences between the means (P-values < 0.05) based on the ANOVA tests performed on this data.² Laboratory #9's result lies on the 0.05 significance level and was not treated as a significant result. Given these considerations, the results of these ANOVAs (with one exception, #6)

²Laboratory #1 should likely be discounted from the other laboratories as the participating subjects were given both the assignment specification and its source code solution to enter during the laboratory session. Furthermore, all students using the *CS1 Sandbox* (regardless of version) might be considered to be at a disadvantage, as they were using the programming environment for the first time. Some of their peers using .NET had previously used Visual C++ 6.0 and therefore had a working knowledge of the Microsoft-style of programming environments/tasks.

Table 6.7: ANOVA Test Results for Mean Closed-Laboratory Scores (laboratories #1 through #13).

Laboratory #	Mean laboratory scores			Results	
	.NET	<i>CS1 Sandbox without subsets</i>	<i>CS1 Sandbox with subsets</i>	F-ratio	P-value
1	99.37	98.30	98.93	3.45	0.03
2	97.65	97.80	98.49	0.52	0.60
3	94.91	92.77	92.28	0.92	0.40
4	97.24	97.68	97.80	0.28	0.76
5	99.81	99.84	99.24	1.77	0.17
6	100.00	96.82	96.83	5.02	0.01
7	100.00	99.84	100.00	1.38	0.25
8	100.00	100.00	100.00	Undefined	0.00
9	100.00	99.52	100.00	3.07	0.05
10	97.66	94.14	94.53	2.72	0.07
11	99.62	99.19	100.00	0.64	0.53
12	88.53	92.14	92.41	1.49	0.23
13	99.12	99.28	98.61	0.20	0.82

indicate that the subjects in the closed-laboratories consistently did no better (and no worse) than their peers no matter which programming environment they used.

The results of a Duncan's multiple range test (used to identify one or more equal pairs) of the significant results (scores from labs #1 and #6) are summarized in Table 6.8. The results are shown in terms of a rank ordering of the sample means with a line drawn under any pairs of means which equality is determined.

Table 6.8: Duncan's multiple range test – closed-laboratory scores from laboratories #1 and #6

Closed-Laboratory #1		
<u>.NET</u>	<i>CS1 Sandbox with</i>	<i>CS1 Sandbox without</i>
99.37	98.30	98.93
97.65	97.80	98.49
94.91	92.77	92.28
97.24	97.68	97.80
99.81	99.84	99.24
100.00	96.82	96.83
100.00	99.84	100.00
100.00	100.00	100.00
100.00	99.52	100.00
97.66	94.14	94.53
99.62	99.19	100.00
88.53	92.14	92.41
99.12	99.28	98.61

Closed-Laboratory #6		
<u>.NET</u>	<i>CS1 Sandbox with</i>	<i>CS1 Sandbox without</i>
99.37	98.30	98.93
97.65	97.80	98.49
94.91	92.77	92.28
97.24	97.68	97.80
99.81	99.84	99.24
100.00	96.82	96.83
100.00	99.84	100.00
100.00	100.00	100.00
100.00	99.52	100.00
97.66	94.14	94.53
99.62	99.19	100.00
88.53	92.14	92.41
99.12	99.28	98.61

Reviewing the graph of the data (see Figure 6.1), the following empirical issues are noted:

1. The across-the-board decline in mean closed-laboratory scores in laboratory #3 is very likely the result of the assignment during that laboratory session. The third laboratory contained the first programming specification without providing any source code to the subjects. Prior to this laboratory, subjects were given laboratory assignments which contained the entire specification and

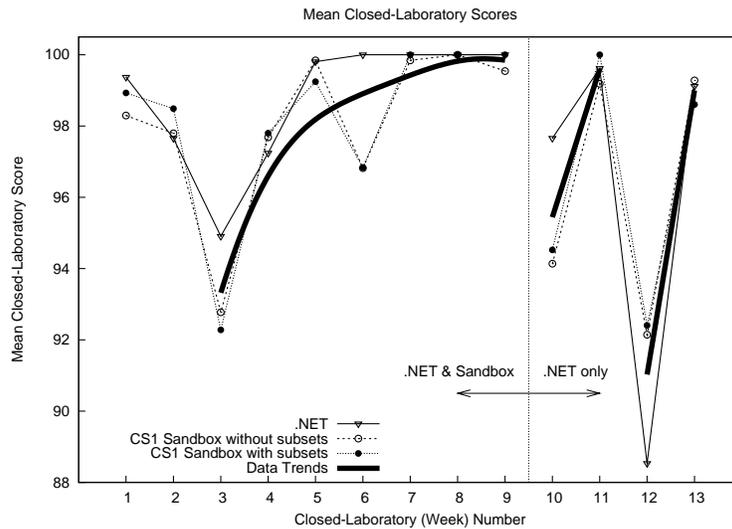


Figure 6.1: Graph of mean closed-laboratory scores (laboratories #1 – #13) with data trends

source code solution (laboratory #1), or the specification with a large percentage of the source code present (laboratory #2). Thus, laboratory #3 was noted for being the first closed-laboratory assignment where subjects were tasked to program without such source code aids.

2. The mean scores of laboratory #10's mean scores are significant for two reasons.

- It was the first laboratory which involved the use of the .NET programming environment for those whom has previously used the *CS1 Sandbox* environment. This author speculates that the effect of moving to a more complex programming environment interface (.NET) may have had some type of negative impact, as the post-project surveys indicate an increase in perceived complexity (discussed further in Section 6.4.5.3)
- Laboratory #10 was the first laboratory in which the subjects were tasked to create, populate, and access an array. Arrays have traditionally³ been a conceptually challenging topic in the CS1044 course for many students.

A decline in the resulting scores in all of the experiment groups was not unexpected by this author based on prior experience managing the closed-laboratory sessions. Note that the subsequent laboratory (#11) the following week saw all three experiment groups' mean scores rise significantly (likely in part due to the experience gained with the use of arrays in the previous week's laboratory).

3. The mean scores of laboratory #12's means scores also exhibited a notable decline. In this laboratory, the subjects were asked to combine both arrays and structures in the assignment to create an array of structures. This also is a laboratory which exercises notions that can be conceptually difficult for some students to implement as it requires the correct application the notion of a nested encapsulating structures.

A decline in the resulting scores in all of the experiment groups was not unexpected based on prior experience by this author managing the closed-laboratories. Note that the subsequent laboratory

³Based on this author's experience as an instructor and teaching assistant for CS1044.

(#13) the following week saw all three experiment groups' mean scores rise significantly (likely in part due to the experience gained with the use of structures in the previous week's laboratory).

Despite the lack of significant results from the ANOVA test performed on this data, there seems to be some coherent trends present in the data (highlighted in Figure 6.1). First, from laboratory #3 through laboratory #9, the overall trend for mean grades is in an upward direction. This trend indicates learning improvements from week to week leading up to the migration for all *CS1 Sandbox* users to the .NET programming environment. Similar smaller learning improvement trends exist between the 10th and 11th laboratories, and 12th and 13th laboratories. The 13th laboratory was an extension of the 12th laboratory, and as such we should expect to see an incremental improvement in the 13th laboratory.

6.3.2 Analysis of Closed-Laboratory Compilation / Execution Attempts

In addition to the mean scores for each laboratory, the number of compilations triggered by each subject was extracted. The mean number of compilations is graphed in Figure 5.2. Table 6.9 redisplayes the collected data and provides the results of the ANOVA tests performed on each laboratory's mean number of compilation attempts by the subjects.

Table 6.9: ANOVA Test Results for Mean Closed-Laboratory Compilations (laboratories #1 through #9).

Laboratory #	Mean number of laboratory compilations		Results	
	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets	F-ratio	P-value
1	3.21	5.60	9.95	0.00
2	20.61	20.45	0.00	0.96
3	45.48	35.28	2.40	0.13
4	27.86	24.54	0.36	0.55
5	20.09	15.63	2.73	0.10
6	32.21	24.05	2.21	0.14
7	18.68	14.93	1.03	0.31
8	23.00	16.12	1.47	0.23
9	14.93	13.81	0.11	0.74

This test seeks to determine if there are unequal means among the groups with respect to the number of compilation and execution attempts to complete the assignment during the laboratory sessions. The number of these attempts provides potential indicators related to a student's performance. First, a higher compilation rate may indicate an elevated frustration level with the language or the programming environment. It is not uncommon to see students (in frustration) attempt to compile a given program multiple times in a rapid fashion without viewing the intervening warning messages (this author terms this behavior as "rapid-fire compiling"). Second, a lower compilation rate may indicate that subjects are able to complete the assigned task with less of a need to re-compile source code (to determine if there are errors present) or to re-execute the program (to determine if logic errors are present).

Admittedly, this is a subjective measure in a sense, as each subject is unique in terms of how they approach software development. However, this is the reason for each we examine the means of these groups of similar subjects. If the means are equal among the groups under analysis, we can say that there seems to be no effect on the outcome of the results (number of compilations/executions). If the results indicated otherwise, we should look to further examine the reason behind the difference in the results.

The original research question was whether subjects who have language subsets applied to their programming environment (*CS1 Sandbox* with subsets) have a statistically significant lower mean number of compilation/execution attempts in completing the closed-laboratory assignment than their counterparts not using subsets (*CS1 Sandbox* without subsets). By executing an ANOVA analysis on this data, we can evaluate the collected data to determine if any significant results have been observed.

The results of these statistical tests indicated that only laboratory #1 showed significance across the means of the two groups. With only one laboratory indicating non-equal means, and the misgivings of counting laboratory #1 expressed in Section 6.3, this author is forced to conclude that the programming environment used had no significant effect upon the mean number of compilations a subject performed while working on a closed-laboratory assignment. In reviewing the graph of this data set (see Figure 6.2),

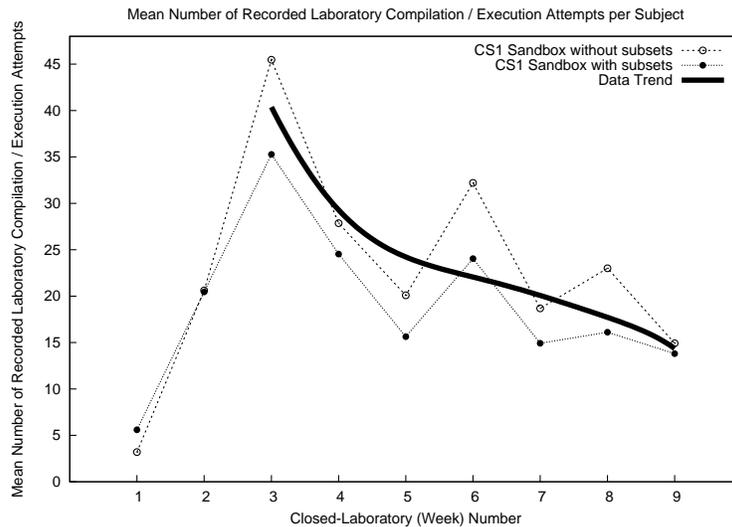


Figure 6.2: Graph of mean number of closed-laboratory compilations (laboratories #1 – #9) with data trend

the following empirical issues are noted:

1. The mean number of compilation/execution attempts from laboratory #3 towers over all the values from other laboratories. This is to be expected as this was the first laboratory in which subjects were using the *CS1 Sandbox* programming environment to author a program without being given the solution or a portion of the source code solution. It is very likely that the acclimation process of developing one's own code for the first time led to additional errors obtained at compile time⁴.
2. The mean number of compilation/execution attempts from laboratory #6 also increases noticeably over the two prior week's laboratories. During the 6th laboratory, subjects received their first

⁴As opposed to previous laboratories in which subjects were given complete or partial solutions to enter.

assignment involving the use of nested loops. A minimal code framework was provided to the subjects (see Appendix D.6), but it only included one of the two required loops. This author suspects that the increased number of compilation/execution attempts during this laboratory session may be in part to an increased number of endless loops encountered by the subjects.

Despite there being only one significant statistical result (laboratory #1) from the execution of the ANOVA analysis, are several significant trends noted from this data. In every laboratory session from the 3rd through the 9th, the subjects who used the *CS1 Sandbox* without subsets had a noticeably higher mean number of compilation/execution attempts compared to those subjects who used the *CS1 Sandbox* with language subsets. Though there is no statistical significance to the data, clearly a trend of lower compilation/execution attempts is present. On the average, the mean difference between the groups in the number of compilations is 22%.

In addition to the lower number of compilation/execution attempts across the board for users of the *CS1 Sandbox* without subsets, the overall trend for all subjects captured in this data set from the point of working without being given source code to start (laboratory #3) is clearly decreasing. This occurs in spite of the fact that from week to week with which the overall complexity of the laboratory assignments increases. The trend slows slightly as the 6th through 9th laboratories are completed, however the trend remains clearly downward. This downward indicator provides confirmation that as the semester progresses and the subjects' use of their programming environment increases, they become more efficient – learning methods and skills to complete the increasingly complex assigned work in a decreasing number of compilation attempts.

6.3.3 Analysis of Closed-Laboratory Compilation / Execution Errors

Another data set under examination by this dissertation is the mean number of compilation/execution errors collected during each laboratory session. Figure 5.3 graphs the mean results for each experiment category, while Table 6.10 provides both the raw mean data values as well as the ANOVA statistical test results on each laboratory's data.

This test seeks to determine if there are unequal means among the groups with respect to the number of compilation and execution errors that arise during the attempts at completing the assignment during the laboratory sessions. As with the number of compilation/execution attempts, a significant result in the mean number of errors has the potential to reflect several salient points about the level of comprehension and success subjects are attaining. Subject who are struggling with the programming language or the environment in which they are operating are likely to produce more errors than their counterparts who are more successful. Frustrated subjects may fail to fully read errors messages which are produced, or neglect to pick up key pointers from those messages and attempt a re-compilation without fully checking their existing source code.

The original research question was whether subjects who have language subsets applied to their programming environment (*CS1 Sandbox* with subsets) have a statistically significant lower mean number of errors in completing the closed-laboratory assignment than their counterparts not using subsets (*CS1 Sandbox* without subsets). By executing an ANOVA analysis on this data, we can evaluate the collected data to determine if any significant results have been observed.

As with the mean number of closed-laboratory compilations (see Section 6.3.2), only the data collected from laboratory #1 proved to be statistically significant (i.e. the means between the groups were unequal). All of the remaining laboratories' data result in statistically equal means. This thereby forces the conclusion that the programming environment used had no significant effect upon the mean number of errors produced while attempting compilation or executions while working on a closed-laboratory assignment.

Table 6.10: ANOVA Test Results for Mean Closed-Laboratory Compilation Errors (laboratories #1 through #9).

Laboratory #	Mean number of laboratory compilation errors		Results	
	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets	F-ratio	P-value
1	1.59	6.93	9.73	0.00
2	14.34	22.26	0.86	0.36
3	72.18	52.86	0.35	0.56
4	45.80	56.49	0.22	0.64
5	18.16	29.29	2.86	0.10
6	21.35	21.38	0.00	1.00
7	30.50	25.30	0.32	0.57
8	58.91	40.02	1.32	0.25
9	25.67	20.02	0.17	0.68

In reviewing the graph of this data set (see Figure 6.3), the following empirical issues are noted:

1. During laboratory #3, users of the *CS1 Sandbox* without subsets had a noticeably larger mean number of compilation/execution errors than users of the *CS1 Sandbox* with subsets. Though the value is not statistically significant (large variances in the data set prevent this from being a significant result), it is most likely due to the fact that this was the first closed-laboratory in which subjects were not given a complete or partial source code solution. This was their first venture into developing a complete solution from scratch.
2. Laboratory #8 was also noted for a spike in the mean number of errors from both *CS1 Sandbox* environments as compared to surrounding weeks. This laboratory session was noted for the requirement to use multiple user-defined functions in the solution. There is no apparent discernible cause as to the reason of the spike in the calculated means.

Another significant trend is very evident (see Figure 6.3) – the decreasing trend in the number of errors received as the laboratory sessions progress. This is a particularly interesting development considering that as the laboratories process (week to week) they generally increase in complexity. Thus, it was not expected to see subjects receiving a decreasing number of errors though it is likely that this may be inter-related to the generally declining number of compilation attempts also observed in the closed-laboratories (see Section 6.3.2).

6.3.4 Closed-Laboratory Summary

Of the statistical tests performed on the three different data sets collected from the closed-laboratory portion of the experiment, four were statistically significant. Of these four significant results, this author suggests discarding three of the results as they are associated with laboratory session #1 – where the students were given the entire specification and solution during the introduction to the laboratory. The

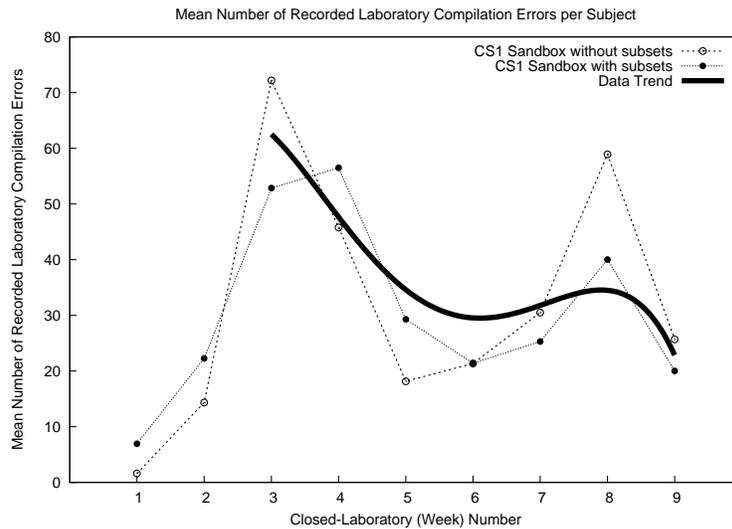


Figure 6.3: Graph of mean number of closed-laboratory compilation errors (laboratories #1 – #9) with data trend

first laboratory, for all intents and purposes, was an exercise in exploring the subject’s ability to use the program development environment. This leaves one statistically significant result.

There are several other points worth mentioning as a conclusion is drawn from the statistical analysis performed on the data sets associated with the closed-laboratories. First, are the nature of the laboratory sessions themselves. Each laboratory session was designed around a particular common goal – to expose the students to language constructs and topics that were covered by the lecture sections earlier in the week. In designing the structure of the laboratories, it was agreed by the supervising teaching assistant (this author) and the supervising faculty member (William McQuain) that the laboratories should be used to foster good programming habits and be used to promote success in programming.

To that end, it was decided that no student who participated in a laboratory session would receive a weekly score less than 70 points. This tactic was used to help students recognize that they were struggling with the material if they received a Curator score less than 70. However, the student’s *recorded* score (used to calculate their class grade) would be a 70. This was also the score we used in this portion of the experiment for the subject. This tactic also helped prevent a struggling student from being demoralized by attending each laboratory session and attempting in good faith to complete the assignment in the time allotted while receiving a poor score. It was believed that such a student would be likely to drop the course, or fail to continue to try to complete the work of the course.

Second, the role of the teaching assistants in each laboratory (both the graduate and undergraduate) likely had an adverse effect upon the results. During each laboratory session, the teaching assistants circulated the room answering questions from the student population as they arose. In many cases, students were given hints, tips, suggestions, and even small fragments of program code to help them complete their assignment. The students were often pushed in the direction of the solution, if it was not coming to them on their own – even after receiving hints from the teaching assistants. The policy of the laboratories was not to give a student any solution, but to assist them in discovering the solution as much as was needed.

Finally, there is a concern over the collected laboratory scores. In many cases, especially the later laboratory sessions in the semester, a large population of the subjects received scores of 100s (refer to the

means from laboratories #7, #8, #9 and #11 which are no more than 1 point from each other). This likely was due to the positive mentoring activities of the teaching assistants during the laboratory sessions.

Such scores are a strong indicator that the data is not normally distributed (i.e. the frequency of scores is distributed among a “normal curve” or “normal probability distribution”⁵ and is symmetrical about its mean μ). Because the laboratory score data appears to be not normally distributed, parametric statistical tests (such as ANOVA tests) are not valid in analysis. This would seem to lead this researcher to using non-parametric tests on the data, however this is not possible either as non-parametric tests rely on the data being continuous and having a probability of ties among the data to be near zero. This is clearly not the case in this data, as there are numerous ties at the 100 score mark among other scores.

A final analysis possibility [70] is that the closed-laboratory score data set be transformed into categorical data. In doing so, each data point would be a yes or no result if the subject attained a 100 score for successfully completing the laboratory assignment. This process was deemed to be an unacceptable translation of the data collected as it does little to assess amount of failure for subjects who did not achieve a perfect score. In light of the mentoring role of the teaching assistants, it was deemed that the underlying laboratory scores are somewhat tainted and the categorical reclassification of the laboratory results was not chosen as an analysis option.

As the result of these three key points from the laboratories:

- the laboratory grading policy,
- the laboratory assistant policy, and
- the lack of normal distribution of scores;

this author concludes that the closed-laboratory data sets should be discounted when drawing any conclusions in this dissertation. Despite these three issues, there remains the presence of an interesting trend discussed previously (that of lower mean number of compilation/execution attempts by those subjects who used the *CS1 Sandbox* with subsets vs. those who used the *CS1 Sandbox* without subsets). This trend will be examined in the out-of-laboratory project results in the next section.

However, the closed-laboratory sessions did provide a clear indication of trends resulting from the continued use of a programming environment as well as the experience gained from weekly assignments structured around the corresponding lectures. These trends include:

1. a general increasing trend of raw score improvement during the laboratories week-to-week,
2. a solidly decreasing trend of the number of compilation and execution attempts in completing the laboratory assignments, and
3. a solidly decreasing trend of the number of compilation and execution errors in completing the laboratory assignments.

6.4 Statistical Evaluation of Out-of-Laboratory Project Results

As described in Section 4.6, the out-of-laboratory project portion of the experiment resulted in the production of several data sets this dissertation needed to examine. This data included the following:

⁵This is frequently known as a bell-shaped curve.

1. the mean scores from the projects,
2. the mean number of compilations by each subject in completing each project,
3. the mean number of errors generated by each subject completing each project,
4. the mean time-on-task by each subject in completing each project, and
5. the post-project surveys.

Of these four data sets, only the first (mean scores) and last (surveys) could include those subjects using the .NET programming environment (recall that the .NET programming environment had no ability to submit data collection results such as source code, errors and the like at the point of compilation/execution). The middle two data sets (number of compilations and the number of errors generated) were only obtainable when using the *CS1 Sandbox* programming environment.

6.4.1 Analysis of Out-of-Laboratory Project Scores

ANOVA statistical tests were run on each of the mean out-of-laboratory project scores. The raw mean scores (graphed in Figure 5.4 and previously presented in Table 5.5) spanned all three experiment groups (.NET, Sandbox without subsets, and Sandbox with subsets) and produced the ANOVA F-ratios and P-values shown in Table 6.11.

Table 6.11: ANOVA Test Results Mean Out-of-Laboratory Project Scores (projects #1 through #5)

Project #	Mean project scores			Results	
	.NET	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets	F-ratio	P-value
1	97.14	98.39	96.95	0.86	0.43
2	98.35	98.05	96.79	1.25	0.29
3	98.56	98.85	94.79	2.19	0.12
4	95.97	98.15	95.69	0.46	0.63
5	97.50	97.34	97.11	0.06	0.94

This statistical test seeks to determine if there are unequal means among the groups with respect to the resulting scores obtained in completing the out-of-laboratory projects. It was hypothesized by this author that subjects using a simplified interface (*CS1 Sandbox* without subsets) have statistically significant higher mean academic scores than their counterparts using an environment with a more complex interface (.NET). Additionally it was hypothesized that subjects who have language subsets applied to their programming environment (*CS1 Sandbox* without subsets) have a statistically significant higher mean academic scores than their counterparts not using subsets (*CS1 Sandbox* without subsets). By executing an ANOVA analysis on this data, we can evaluate if any significant results have been observed.

Each of the tests resulted in P-values greater than 0.05, resulting in the rejection of one of the proposal research questions (subjects using a programming environment with language subsets will have higher mean academic scores than their peers who do not). Therefore, it is reluctantly concluded that there was

no significant impact of the programming environment upon the subject's score performance in the out-of-laboratory take-home projects.

Visual inspection of the graph of the mean project scores (see Figure 6.4) indicates two interesting points. First, for each of the projects, subjects who used the *CS1 Sandbox* with subsets had lower means than their peers in the other two experiment groups (.NET, and *CS1 Sandbox* without subsets). This trend seems to indicate that student performance may have even been slightly hindered by the use of the *CS1 Sandbox* with subsets.

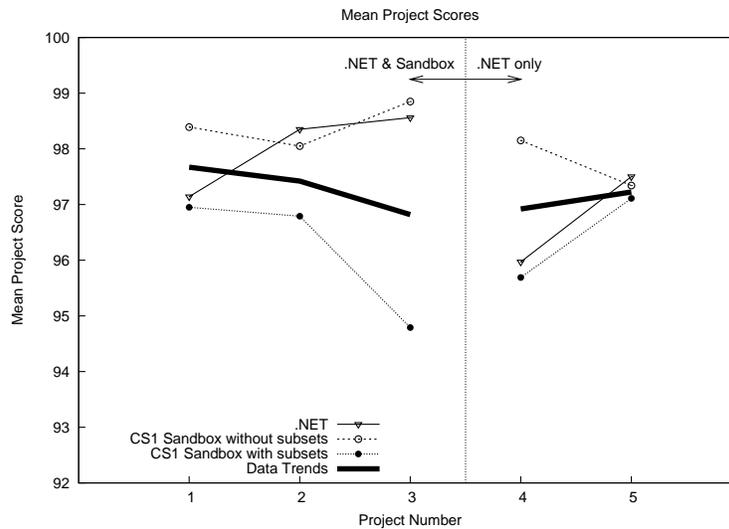


Figure 6.4: Graph of mean out-of-laboratory project scores (projects #1 – #5) with data trends

It is not discernible why this trend is present, as it seems to be an anomaly – no such trend exists in the closed-laboratory results, and the software used by this experiment group is identical to the *CS1 Sandbox* without subsets group with only the application of language subsets. Any technical problems experienced in the programming environment that might hinder compilation or completion of the out-of-laboratory projects would be seen in both of the experiment groups that used the *CS1 Sandbox*.

Second, while peer experiment groups (.NET, and *CS1 Sandbox* without subsets) saw slight increases in their mean out-of-laboratory programming project score for project #3, those subjects in the *CS1 Sandbox* with subsets experiment group saw a sharp decline in mean scores. Following project #3, the reverse was true, subjects using the *CS1 Sandbox* with subsets rebounded with higher mean scores on project #4, while their peers' mean scores declined.

This behavior on the part of the *CS1 Sandbox* with subsets group is the impact of one subject who obtained a raw score of zero on the third project. This subject submitted 5 candidate source code solutions to the Curator (the maximum permitted for this project) and on each occasion received a zero score. Twice, the subject's submission would not compile (1st and 5th submission), yielding errors related to the redefinition of an existing function. The other three submissions compiled without error, but failed to produce the required output file and were thus scored a with a zero. The overall zero score on the project was the only zero score received by any subject for this project.

Only one other subject scored a zero grade in any other out-of-laboratory programming project (a subject using .NET on the 4th project). Without the zero score factored into the 3rd project's means, the mean

score for the *CS1 Sandbox* with subset group would have risen to a value of 97.29, eliminating the sharp decline for the 3rd project, and producing a trend-matching decline on the 4th project. The graph of the mean project scores is replotted in Figure 6.5 with these zero project scores removed for comparison against Figure 6.4.

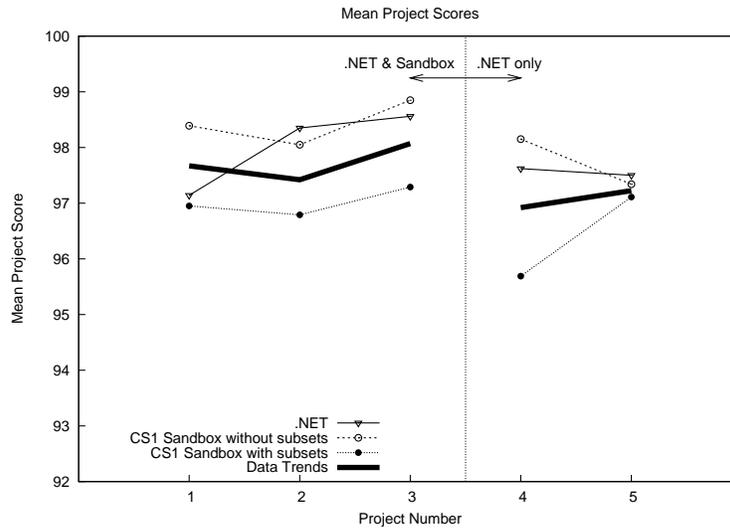


Figure 6.5: Graph of mean out-of-laboratory project scores (projects #1 – #5) with data trends (with both zero project scores removed)

One final point worth mentioning with regards to the observed mean scores is the overall trends indicated in Figure 6.4. Both trends, despite the sharp decline in project #3, are nearly flat. Without the zero score in the third project, the trend line for the first three projects would all but level out as the result of the higher mean for the third project.

6.4.2 Analysis of Out-of-Laboratory Project Compilation / Execution Attempts

The mean number of compilations recorded during the completion of the out-of-laboratory projects by the subjects has been previously displayed in Table 5.6. Table 6.12 provides the results of the repeated measures test performed on this data set.

The repeated measures test seeks to determine if there are unequal means among the groups with respect to the number of compilation and execution attempts required to complete the out-of-laboratory projects. It was hypothesized by this author that subjects who have language subsets applied to their programming environment (*CS1 Sandbox* with subsets) have a statistically significant lower mean number of compilation/execution attempts in completing the assignment than their counterparts not using subsets (*CS1 Sandbox* without subsets). By executing an analysis on this data, we can evaluate the collected data to determine if any significant results have been observed.

With a resulting P-value of 0.92 from the repeated measures test, the original hypothesis is rejected (that users of a programming environment supporting language subsets will have a statistically significantly fewer

Table 6.12: Repeated Measures Test Results for Mean Out-Of-Laboratory Compilations

Source	DF	Type III SS	Mean Square	F-value	P-value
Programming Environment	1	97.90	97.90	0.01	0.92
Error	62	548951.43	8854.06		

number of compilation/execution attempts). Therefore, it is concluded that there was no significant impact of the programming environment upon the subject's compilation count in the out-of-laboratory projects.

Despite the lack of significant results for this data set, there are strong empirical indications of a positive trend in the data which may support the notion that subjects using the programming environment with language subsets require fewer compilations to complete their projects than do their peers using the programming environment without language subsets.

Examining the graph of the means from this data set again (see Figure 6.6), it is noted that that for each project assigned, those subjects using the *CS1 Sandbox* with subsets consistently required less compilations than those subjects using the *CS1 Sandbox* without subsets. This consistent reduction ranged from 10% to 33% across the three projects, with a mean 18% reduction. This is distinctly similar to the reduction in the number of compilation/execution attempts noted in the closed-laboratories (see Section 6.3.2) where a 22% reduction was observed by those using the *CS1 Sandbox* with subsets.

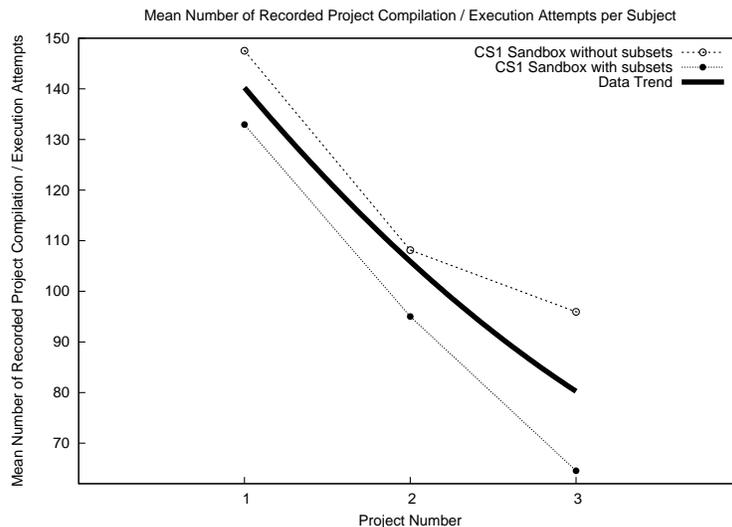


Figure 6.6: Graph of mean number of out-of-laboratory project compilations (projects #1 – #3) with data trend

One interesting point of note is the direction of the graph's plots. Clearly, the trend in the data is for fewer and fewer compilations as each project is completed. This seems to be a surprising result despite the fact that each subsequent project was more complex and involved than its predecessor. Common sense would seem to tell us that additional compilations would be required to complete the projects if their complexity increased. This author hypothesizes that this interesting trend is due to student acclimation to the programming environment – as the subjects continued to work with the environment (and became better

overall programmers), their efficiency increased resulting in fewer compilations to accomplish their programming tasks (i.e. the subjects were learning how to use the programming environment).

An additional explanation for the downward trend is that the second project builds upon the first, and the third project builds upon the second. Common sense would indicate that not only is it likely that the subjects are becoming more acclimated to the programming environment, but the subjects may be realizing that fewer compilation attempts are required to complete the subsequent projects since a portion of the existing code is in fact being reused.

As expected, the mean number of compilation and execution attempts ranged from 4.6 to 3.2 times larger than the closed-laboratory compilation / execution attempts (excluding laboratories #1 and #2). This is not wholly unexpected as the out-of-laboratory projects were certainly more complex than the closed-laboratory assignments, and the take-home projects were likely to span a longer time frame than the laboratory programs.

6.4.3 Analysis of Out-of-Laboratory Project Compilation / Execution Errors

Another series of data collected during the experiment was the out-of-laboratory project error rates. This data represents the total number of errors each subject accrued while completing their take-home programming projects. Such errors were the result of compilation attempts (syntax, semantic, or component disabled (subset) messages), as well as interpretation errors (run-time errors). The mean number of compilation / execution errors recorded during the completion of the out-of-laboratory projects by the subjects has been previously shown in Table 5.7. Figure 6.7 redisplay the graph of the data set and indicates the data's trend, while Table 6.13 provides the results of the repeated measures tests performed on the data set.

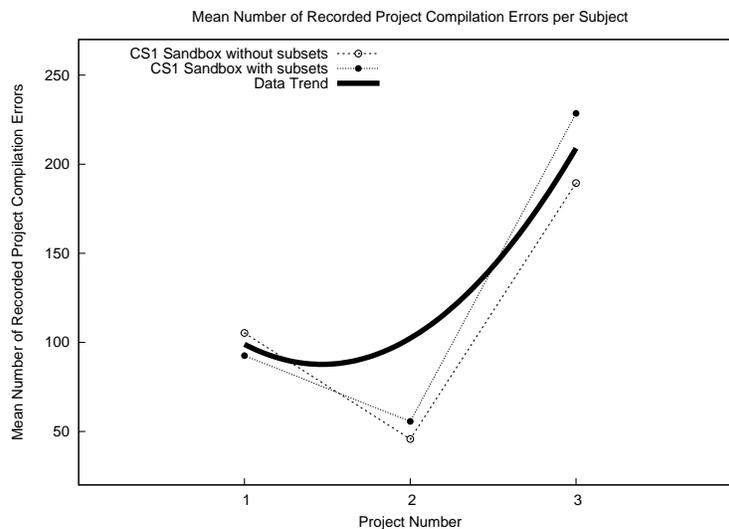


Figure 6.7: Graph of mean number of out-of-laboratory project compilation errors (projects #1 – #3) with data trend

This statistical test seeks to determine if there are unequal means among the groups with respect to the number of compilation and execution errors that arise during the attempts to complete the

Table 6.13: Repeated Measures Test Results for Mean Out-Of-Laboratory Compilation Errors

Source	DF	Type III SS	Mean Square	F-value	P-value
Programming Environment	1	77370.05	77370.05	1.36	0.25
Errors	62	3531973.62	56967.32		

out-of-laboratory projects. As with the number of compilation/execution attempts, a significant result in the mean number of errors has the potential to reflect several salient points about the level of comprehension and success subjects are attaining. Subject who are struggling with the programming language or the environment in which they are operating are likely to produce more errors than their counterparts who are more successful. Frustrated subjects may fail to fully read errors messages which are produced, or neglect to pick up key pointers from those messages and attempt a re-compilation without fully checking their existing source code.

It was hypothesized by this author that subjects who have language subsets applied to their programming environment (*CS1 Sandbox* with subsets) would have a statistically significant lower mean number of errors in completing the out-of-laboratory projects than their counterparts not using subsets (*CS1 Sandbox* without subsets). By executing a repeated measures analysis of variation on this data, we can evaluate the collected data to determine if any significant results have been observed.

With a resulting P-value of 0.25 from the repeated measures test, this author rejects the original hypothesis that the means would be statistically lower for those users of a programming environment which supports language subsetting. Clearly, examination of the graph of the means (see Figure 6.7) reflects a slightly higher mean number of errors for subjects using the *CS1 Sandbox* with subsets than with those who used the *CS1 Sandbox* without language subsets.

If one compares the mean number of compilation/execution attempts for each project against the mean number of errors obtained, and interesting observation comes to light. For the first programming project, the mean number of errors is roughly 70% less than the mean number of compilation/execution attempts. For the second project, this value climbs slightly – there are roughly half as many errors as compilation/execution attempts. However, during the third project, the upward trend continues – albeit much more sharply. Here, the number of errors is 250% greater than the number of compilation/execution attempts. Clearly, there is a sharp increase in the number of errors coming from the third project (see Figure 6.7). This is a surprising result since it was noted in the previous section that the compilation/execution rate was steadily decreasing and the projects built upon each other.

This author speculates that the cause may lie within the programming environment. Project #3 dealt with the implementation of several user-defined functions, and there was an implementation issue in the *CS1 Sandbox* programming environment related to the passing of parameters across user-defined functions. The issue was addressed and corrected prior to the completion of the third out-of-laboratory programming project's due date, but it may have contributed to the elevated error rate by spawning a larger number of early errors. Similarly, the corresponding contemporaneous laboratory that dealt with multiple user-defined functions (laboratory #8) saw sharp increases in its error rate (see Figure 6.3).

6.4.4 Analysis of Out-of-Laboratory Project Time-On-Task

The time-on-task data for the out-of-laboratory projects represents the mean time-on-task for each experiment group. The value was calculated by locating the time-on-task for each subject from their first recorded compilation attempt to their final compilation/execution attempt. The resulting time difference

(in seconds) was recorded as the subject’s time-on-task. In most cases, this value spanned several days, and included “down” time such as when the subjects were sleeping or otherwise in classes.

This statistical test seeks to determine if there are unequal means among the groups with respect to the mean time-on-task needed by the subjects to complete the out-of-laboratory projects. It was hypothesized by this author that subjects using a simplified interface (*CS1 Sandbox* with subsets) have statistically significant lower time-on-task values than their counterparts not applying language subsets to their environment (*CS1 Sandbox* without subsets). Given a significant result from this test we can state that language subsets help the subjects complete their tasks more quickly. Additional examination of the data could then be undertaken to determine how many programming sessions the task was completed over.⁶ The mean time-on-task recorded during the completion of the out-of-laboratory projects by the subjects has been previously shown in Table 5.8. Table 6.14 provides the results of the repeated measures tests performed on the data set.

Table 6.14: Repeated Measures Test Results for Mean Out-Of-Laboratory Time-On-Task for Programming Projects

Source	DF	Type III SS	Mean Square	F-value	P-value
Programming Environment	1	82648129142.00	82648129142.00	0.70	0.41
Errors	62	7.33	118161968064.00		

With a resulting P-value of 0.41 from the repeated measures test, this author rejects the original hypothesis that the means would be statistically lower for those users of a programming environment which supports language subsetting. Thus, this author concludes that the introduction of subsets did not significantly help the subjects who used the *CS1 Sandbox* with subsets complete their projects in a more timely fashion.

One trend from the data set that did emerge (see Figure 6.8) is that in two of the three projects, subjects using the *CS1 Sandbox* with subsets had slightly lessor mean time-on-task measures than their peers who used the *CS1 Sandbox* without subsets.

Similar to the results of obtained when examining the number of compilations required to complete the programming exercise, an interesting trend is noted regarding the time-on-task of the programming projects – decreasing time-on-task. As each project is accomplished, the mean time-on-task decreases.

Despite the fact that project #1 was open for completion for a longer period of time than both projects #2 and #3 (23 days vs. 14 days for projects #2 and #3), the trend is clearly that less time is required for each project – despite the projects becoming increasingly more complex. Table 6.15 reiterates this trend when the mean time-on-task is viewed as a percentage of the overall available time to complete the project, rather than just as the overall time from end-to-end.

6.4.5 Analysis of Out-of-Laboratory Post-Project Survey Results

Both of two self-reported, self-estimated results on the post-project surveys (see Appendix F) were analyzed using standard ANOVA statistical tests to determine if the means among the experiment groups were equal. For each of the rating-type questions in the post-project surveys, a Kruskal-Wallis analysis (see Section 6.1.3) was performed to determine if the mean ranking values were statistically different between

⁶This activity is planned as a future research endeavor of the data collected.

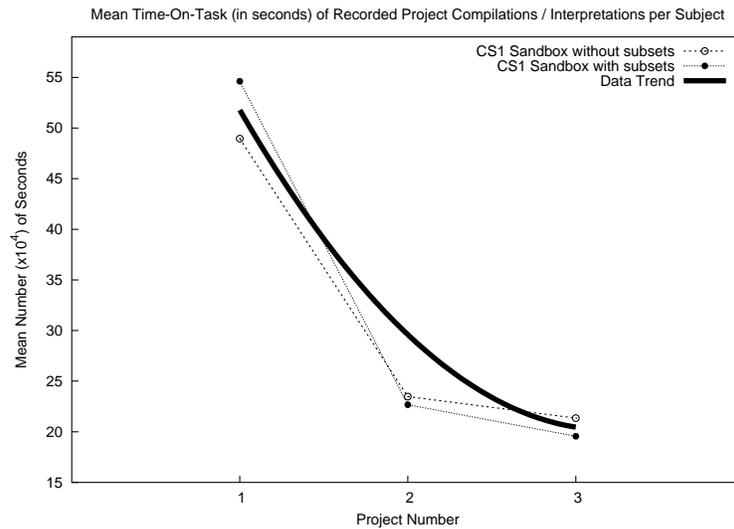


Figure 6.8: Graph of mean time-on-task (in seconds) of out-of-laboratory projects (projects #1 – #3) with data trend

the experiment groups. With seven principal questions of interest issued to the subjects on five separate occasions, this resulted in the execution of ten ANOVA tests for the self-reported data (see Tables 6.16 and 6.17) and 25 separate Kruskal-Wallis statistical tests for the rating-type data (see Tables 6.18 through 6.25). Analysis of the data sets obtained from each question appears in the sections that follow.

6.4.5.1 Post-Project Survey Question: Self-Reported Number of Days on Task

Post-project survey question #7 (see Appendix F) intended to obtain a measure of the self-reported number of days on task to complete each out-of-laboratory programming assignment. The primary reason for the existence of this question was to obtain some level of measure of the requested data for .NET users, and be therefore able to undertake a comparison of said values against the users of the *CS1 Sandbox*

Table 6.15: Mean Out-Of-Laboratory Time-On-Task Percentage for Programming Projects (projects #1 through #3).

Survey #	Mean % of project time-on-task	
	% of total time allowed (without subsets)	% of total time allowed (with subsets)
#1	24%	26%
#2	19%	19%
#3	18%	16%

programming environment.

As Table 6.16 reflects, each of the five resulting P-values are above the 0.05 significance level (though the first is very close to the significance level). Thus, the null statistical test hypothesis (the means are equal) is accepted in each case (all five tests), and it can be concluded that based on the self-reported number of days on task, no experiment group out-performed another group.

Table 6.16: Mean self-reported values from the post-project survey question: *How many days did you work on the project?* (Note: only count a day once when you worked on the project one or more times)

Survey #	Mean reported days on task			Results	
	.NET	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets	F-ratio	P-value
#1	2.78	3.53	3.11	2.72	0.07
#2	2.29	2.44	2.11	0.92	0.40
#3	2.48	2.21	2.13	0.88	0.42
#4	3.38	2.91	2.97	1.31	0.27
#5	4.04	4.83	3.60	2.34	0.10

Two trends clearly emerge from the data captured in this portion of the experiment (see Figure 6.9). The first is that over the first three projects, subjects self-reported a declining amount of time-on-task to complete the projects. This trend corresponds with the data captured automatically by use of the *CS1 Sandbox* environment and thus validates the veracity of the self-reported values.

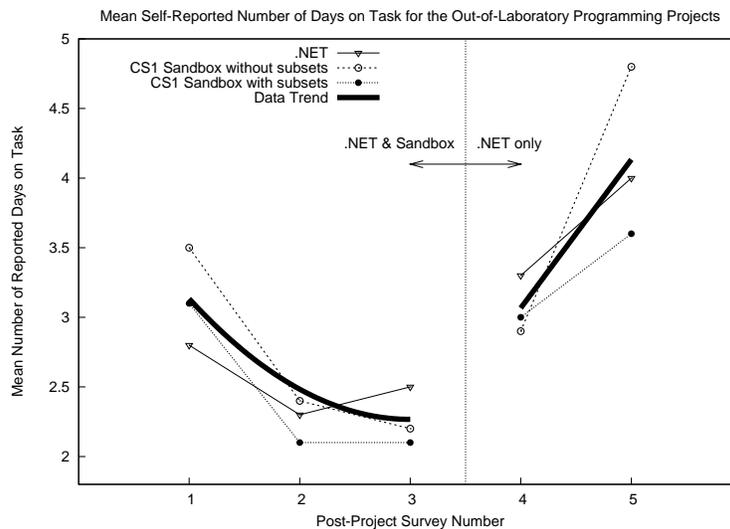


Figure 6.9: Graph of mean self-reported days on task with data trend

The second trend is a sharply increasing mean amount of time-on-task (albeit from three to four days to complete the project) over the fourth and fifth out-of-laboratory programming projects. An elevated value was expected on the fourth project as it (unlike the second and third projects) did not build on the previous projects (and was found to be rather complex). Additionally, migration to the .NET programming environment by the users of the *CS1 Sandbox*

However, the final project's mean self-reported time-on-task comes as a surprise to this researcher. The increase comes despite the fifth project being an extension of the first three projects and therefore it was expected that similar behavior (decreasing trend of the number of days-on-task) would be observed.

6.4.5.2 Post-Project Survey Question: Self-Reported Number of Compilations / Executions

Post-project survey questions #8 and #9 (see Appendix F) was designed to obtain a measure of the self-reported number of compilations and executions for each project undertaken. The primary reason for the existence of this question was to obtain some level of measure for the requested data for .NET users, and be therefore able to undertake a comparison of said values against the users of the *CS1 Sandbox* programming environment. As a side effect, with this data, it is possible to compare the actual values obtained electronically against the self-reported values for the *CS Sandbox* users.

As Table 6.17 reflects, each of the five resulting P-values are above the 0.05 significance level. Thus, the null statistical test hypothesis (the means are equal) is accepted in each case (all five tests), and it can be concluded that based on the self-reported values of the number of compilations and executions, no experiment group out-performed another group.

Table 6.17: Mean self-reported values from the post-project survey questions: *How many attempts at compilation did you perform while completing the project? (please quantify, don't enter "some")* and *How many attempts at execution did you perform in completing the project? (please quantify, don't enter "some")*

Survey #	Mean reported # of compilations & executions			Results	
	.NET	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets	F-ratio	P-value
#1	76.3	66.6	46.8	2.47	0.09
#2	69.4	73.7	68.1	0.07	0.93
#3	63.0	53.7	57.5	0.37	0.69
#4	136.0	115.6	129.6	0.23	0.80
#5	172.2	160.5	156.2	0.08	0.92

Two interesting trends emerged from this self-reported, self-estimated data. First, the mean number of self-reported compilation/execution attempts is essentially flat (see Figure 6.10) for the first three out-of-laboratory projects. This trend alone sharply contrasts with the clearly downward trend collected from users of the *CS1 Sandbox* over the same time period (see Figure 6.6). The second trend indicates a consistently rising mean number of compilation/execution attempts required to complete the fourth and fifth out-of-laboratory programming projects.

The clearly interesting observation from this data is the measure of perception of "work" (count of compilations and executions) on the part of the subjects. If one compares their self-reported values from

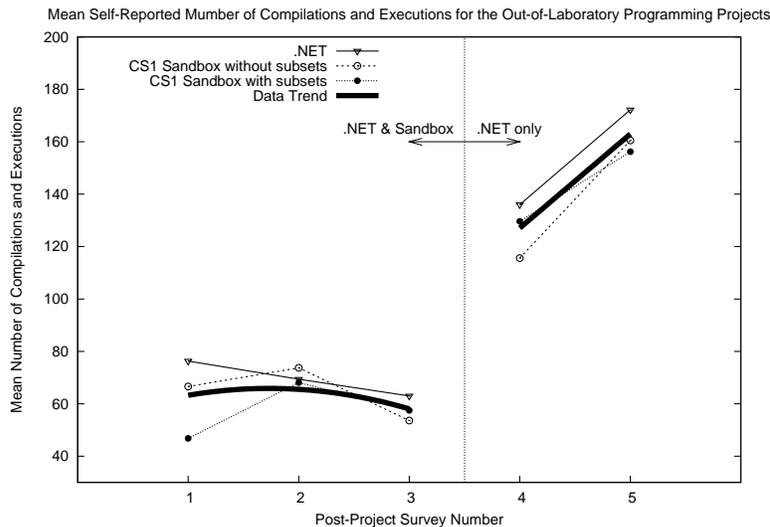


Figure 6.10: Graph of mean self-reported number of compilation/execution attempts with data trend

the first three projects with the actual results obtained through the data collection process over the same time period, it is observed that the subject's perception of their "work" is roughly the same for all three projects (roughly 60-60 per project).

These values contrast sharply with the automatically collected data (shown in Figure 6.6). Not only does the automatically collected data indicate a clearly decreasing mean number of compilations across the three projects, but the number of actual (automatically collected) values is between 1.5 and 2.5 times more than they reported. Clearly, the subjects seem to have a perception problem as it relates to the amount of work they are performing. That is, assuming the self-reported numbers are not intentionally deflated by the subjects (in an attempt to gain favor with this researcher), the subjects seem to be performing nearly twice as much "work" as they realize.

6.4.5.3 Post-Project Survey Question: Rating Start-Up Experience

Post-project survey question #3 (see Appendix F) was designed to obtain a measure of perception of difficulty with respect to starting a new project or program in the subject's programming environment at each stage of the experiment.

Each of the means from the three groups during the first three surveys (following the first three out-of-laboratory projects) were nearly identical (within 0.2 points of each other; see Table 6.18) – at the "Easy" rating. During the survey periods, subjects of the .NET programming environment continued to rate their perceived difficulty level consistently. Results of the Kruskal-Wallis tests on the means from these first three projects (P-values of 0.49, 0.11, and 0.39 respectively) showed no statistical difference in the means.

As expected, as users of the *CS1 Sandbox* (both with and without subsets) migrated to the .NET programming environment following the third programming project, their reported mean ratings of difficulty in starting a new project or program jumped noticeably (see Figure 6.11). The resulting P-values (0.00, and 0.00 for post-project surveys #4 and #5) confirm that the means are not statistically equal.

Table 6.18: Kruskal-Wallis analysis results from the post-project survey question: *How would you categorize your experience in starting a new project/program in the programming environment you used?*

Survey #	Mean rating			P-value
	.NET	CS1 Sandbox without subsets	CS1 Sandbox with subsets	
#1	2.0	1.8	2.0	0.49
#2	1.6	1.8	1.8	0.11
#3	1.8	2.0	1.9	0.39
#4	1.7	2.5	2.9	0.00
#5	1.7	2.2	2.5	0.00

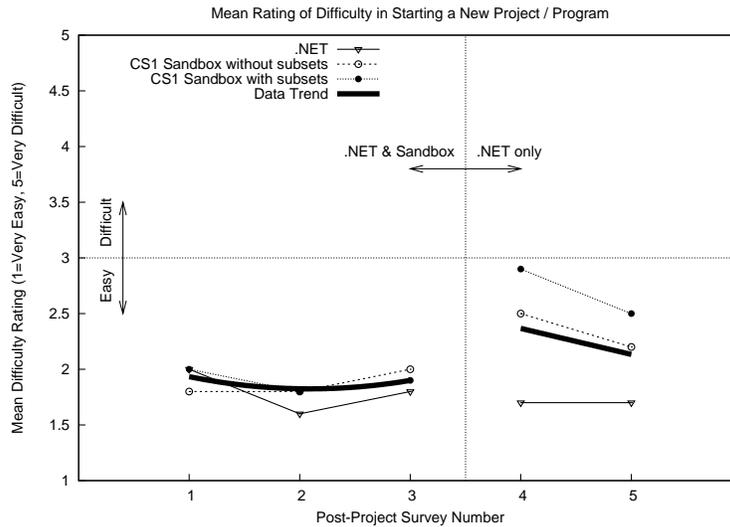


Figure 6.11: Graph of mean survey ratings of difficulty in starting a new project/program with data trends

Further statistical analysis on the results from post-project surveys #4 and #5 was conducted to determine which of the experiment groups differ. This analysis was achieved by executing pair-wise Kruskal-Wallis tests on each of the three pairs of possible combinations of the data. As reflected in Table 6.19, all three of the groups from the post-project #4 survey were found to be different (statistically not equal), while the .NET group was found to differ from the CS1 Sandbox groups in the results from the post-project #5 survey.

This reaction is considered to be the result of the increased complexity of the interface in the .NET programming environment, particularly in the project creation/management interface that confronts users each time the environment is started. Despite using the .NET environment for projects #4 and #5, the subjects who previously used the CS1 Sandbox programming environment continued to report elevated

Table 6.19: Pair-wise Kruskal-Wallis analysis results from post-project surveys #4 & #5 – question #3

	Post-Project Survey P-values	
	#4	#5
.NET vs. <i>CS1 Sandbox</i> without	0.00	0.00
.NET vs. <i>CS1 Sandbox</i> with	0.00	0.00
<i>CS1 Sandbox</i> with vs. <i>CS1 Sandbox</i> without	0.00	0.11

difficulty ratings – noticeably higher than their peers who has worked with .NET all semester long. This author finds it interesting to note that perceived difficulty in working with .NET is significantly higher for *CS1 Sandbox* users in post-project survey #4 (2.5 and 2.9) than for the .NET users in post-project survey #1 (2.0). That is, *CS1 Sandbox* users had higher initial reactions to working with the .NET programming environment (once they migrated to it) than did users from the .NET experiment group.

This researcher hypothesizes that the elevated initial reactions were attributable to the subjects comparing their .NET experience in starting a program/project with their experiences in working with the *CS1 Sandbox*. Common sense would seem to indicate that when migrating from an environment that contains only a handful of menus and buttons (GUI controls) to an environment with considerably more controls users would find this more “complex”. Add to the increased complexity the requirement to utilize project management tools to manage a single-file source code program, the results of this survey are not surprising.

The implication here is that the trends seem to indicate that novice programmers find they have less difficulty in getting started programming with a “simplified” environment. However, those that did not know any better (see the .NET subjects’ responses) had little difficulty in coping with the complexity of starting a new project.

6.4.5.4 Post-Project Survey Question: Rating Perception of Difficulty Entering / Debugging Source Code

Post-project survey question #4 (see Appendix F) was designed to obtain a measure of perception of difficulty with respect to entering and debugging program source code in the subject’s programming environment at each stage of the experiment.

Each of the means from the three groups during the first three surveys (following the first three out-of-laboratory projects) diverged in an interesting pattern (see Table 6.20 and Figure 6.12). The *CS1 Sandbox* users reported higher perceived difficulty in entering and debugging their source code than their .NET peers following the first programming project. However, as the subjects completed projects #2 and #3, the users of the *CS1 Sandbox* began to report increased difficulty in performing their code entry/debugging tasks, while .NET users reported a slightly easier time with their environment.

This author attributes this behavior to two possible sources. The first being *CS1 Sandbox* programming environment flaws occurring during the third programming project (related to the execution of user-defined functions and parameter passing). Second, the *CS1 Sandbox* programming environment contained no tools supporting the debugging of programs (i.e. there was no debugger where student could step through the program and monitor the changing of named memory locations associated with values [variables]). Although little to no official coverage of the debugging topic is presented in the course (a tutorial on how to use the .NET debugger is provided as an appendix to the course note pack), the subjects who used the *CS1 Sandbox* environment clearly found it more difficult to debug their programs without such an aid.

Table 6.20: Kruskal-Wallis analysis results from the post-project survey question: *How would you categorize the difficulty in entering code and debugging syntax or semantic bugs in your project solution using this programming environment?*

Survey #	Mean rating			P-value
	.NET	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets	
#1	2.2	2.7	2.5	0.01
#2	2.1	2.7	2.5	0.00
#3	2.1	2.9	2.7	0.00
#4	2.2	2.4	2.3	0.72
#5	2.1	2.1	2.3	0.44

Another possible factor to the elevated mean ratings for *CS1 Sandbox* users may be due to inconsistent syntax color highlighting. Several subjects noted that the highlighting disabled itself without warning (or incorrectly colored the source code) and the subjects reported this to be a distraction. Their comments on this issue can be seen in Section 5.2.5.3.

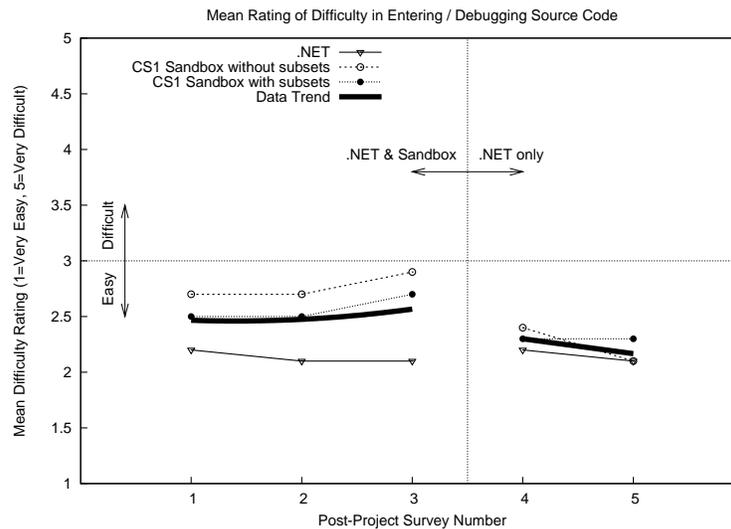


Figure 6.12: Graph of mean survey ratings of difficulty in entering / debugging source code with data trends

As the survey progressed, subjects of the .NET programming environment continued to rate their perceived difficulty level consistently as “easy”. Conversely, as the subjects who used the *CS1 Sandbox* migrated to the .NET programming environment, they reported less difficult experiences in entering and debugging their source code in the .NET programming environment. P-values from the statistical analysis of the survey data following projects #4 and #5 yield equivalent means between the experiment groups.

Further statistical analysis on the results from post-project surveys #1, #2, and #3 was conducted to determine which of the experiment groups differ. This analysis was achieved by executing pair-wise Kruskal-Wallis tests on each of the three pairs of possible combinations of the data. As reflected in Table 6.21, only the .NET group differed from the *CS1 Sandbox* without subsets group in post-project survey #1. In both post-project surveys #2 and #3, the .NET group had a statistically significant advantage over both of the remaining groups.

Table 6.21: Pair-wise Kruskal-Wallis analysis results from post-project surveys #1, #2, and #3 – question #4

	Post-Project Survey P-values		
	#1	#2	#3
.NET vs. <i>CS1 Sandbox</i> without	0.00	0.00	0.00
.NET vs. <i>CS1 Sandbox</i> with	0.11	0.01	0.00
<i>CS1 Sandbox</i> with vs. <i>CS1 Sandbox</i> without	0.30	0.27	0.64

One interesting observation emerging from the data is that both trends (before and after the migration to .NET by the users of the *CS1 Sandbox* programming environment) seemed to be relatively stable and consistent. The pre-migration trend (projects #1, #2, and #3) is slightly “more difficult” than the post-migration trend, as it takes into account the elevated mean ratings obtained from users of the *CS1 Sandbox* programming environment.

6.4.5.5 Post-Project Survey Question: Rating Perception of Testing Program Solutions

Post-project survey question #5 (see Appendix F) was designed to obtain a measure of perception of difficulty with respect to testing program solutions in the subject’s programming environment at each stage of the experiment.

Similar to the mean responses from survey question #4 (see Section 6.4.5.4) regarding code entry and debugging, the means from the *CS1 Sandbox* users and .NET users regarding testing diverged during the first three surveys (following the first three out-of-laboratory projects) (see Table 6.22 and Figure 6.13). The *CS1 Sandbox* users reported higher perceived difficulty in testing their source code than their .NET peers following the first programming project.

Additionally, as the subjects completed projects #2 and #3, the users of the *CS1 Sandbox* began to report increased difficulty in testing their programs, while .NET users reported an easier time with their environment. This author attributes this behavior to the limited testing tools available in the *CS1 Sandbox* environment. The *CS1 Sandbox*, unlike the .NET programming environment, lacked a debugger tool. P-values from the Kruskal-Wallis tests of the survey data following these projects confirm unequal means from the second and third surveys.

Following the migration from the *CS1 Sandbox* to the .NET programming environment (following the completion of the third out-of-laboratory programming project), former *CS1 Sandbox* users reported a noticeably less difficult experience in testing their programs (projects #4 and #5), while their .NET-using peers reported a slight increase in difficulty in testing their programs. Mean values for projects #4 and #5 nearly converged in the survey (confirmed by the Kruskal-Wallis P-values of 0.70 and 0.21) over the final two surveys tested.

Further statistical analysis on the results from post-project surveys #2 and #3 was conducted to determine which of the experiment groups differ. This analysis was achieved by executing pair-wise

Table 6.22: Kruskal-Wallis analysis results from the post-project survey question: *How would you categorize the difficulty in testing your project solution using this programming environment?*

Survey #	Mean rating			P-value
	.NET	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets	
#1	2.1	2.3	2.4	0.31
#2	1.9	2.5	2.4	0.00
#3	1.9	2.8	2.6	0.00
#4	2.2	2.1	2.3	0.70
#5	2.0	2.2	2.3	0.21

Kruskal-Wallis tests on each of the three pairs of possible combinations of the data. As reflected in Table 6.23, the .NET group was confirmed to have statistically significant lower means than those from the *CS1 Sandbox* groups in both post-project surveys.

Table 6.23: Pair-wise Kruskal-Wallis analysis results from post-project surveys #2 and #3 – question #5

	Post-Project Survey P-values	
	#2	#3
.NET vs. <i>CS1 Sandbox</i> without	0.00	0.00
.NET vs. <i>CS1 Sandbox</i> with	0.00	0.00
<i>CS1 Sandbox</i> with vs. <i>CS1 Sandbox</i> without	0.66	0.52

These results for .NET confirm that the debugging tools present in the programming environment provide a distinct advantage to the novice users – contrary to the original hypothesis that such tools are unnecessary for novice programmers. Most surprising to this researcher is this result comes about despite the fact that the course syllabus does not include coverage on how to effectively use these tools to quickly and efficiently locate and correct errors in programs – the students are essentially left to explore these tools on their own!

Most interesting among this data is the trends observed (see Figure 6.13). They are nearly identical to those observed in the previous survey question – rating the difficulty entering and debugging the projects (see Figure 6.12). A possible cause for the similarity may be the the subjects simply do not understand (yet) the difference between debugging and testing of one’s program. In their mind, the act of debugging and testing are very likely the same as the subtle difference had not been covered by the course materials up to this point.

Viewing the results from this survey question with the last seems to indicate that the simplified program development environment is likely best suited for assisting the novice programmers with only the earliest (and therefore most simple) programs. Thus, a conclusion from this finding is that such a simplified system should only be used very early in the course, perhaps less so than the length of time used during the course

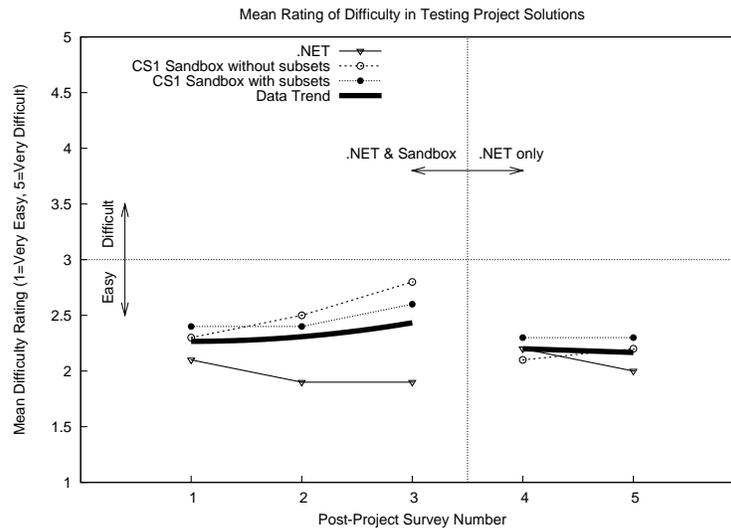


Figure 6.13: Graph of mean survey ratings of difficulty in testing project solution with data trends

of this experiment⁷.

6.4.5.6 Post-Project Survey Question: Rating Perception of Difficulty of the Project(s)

Post-project survey question #6 (see Appendix F) was designed to obtain a measure of perception of difficulty of the programming projects by the users. Kruskal-Wallis tests performed on all five data sets obtained by this question in the survey resulted in P-values greater than 0.05 (see Table 6.24). Considering the graph of the means from this data set (see Figure 6.14), these results are not unexpected as each of the means from the three experiment groups tracked very closely together. This would indicate that subjects from each of the groups held similar perceptions of all five of the programming projects, and likely speaks to the veracity of their answers.

Out-of-laboratory programming project #4 is particularly significant for its extreme spike in perceived difficulty. This is likely due to the nature of the assigned task – simulated cryptography. The output required to obtain a respectable score from the Curator grading system was noted to be rather picky for this project. This author recalls several occasions of students complaining in the closed-laboratory sessions and on the web-based class discussion forum of the increased output difficulties experienced by the subjects.

The lack of significant differences among the means of the experiments' groups speaks toward the perception of the task. Essentially, each experiment group had the same mean rating of the difficulty of the project. This implies that despite the programming environment in use by the subjects, they universally held the same opinions of the amount of work required to complete the project.

⁷The Department of Computer Science at Virginia Tech is changing their CS1 syllabus this coming fall semester (2003) to use a simplified program development environment (BlueJ) in conjunction with a pedagogical sandbox (KarelJRobot) in support of changing the CS1 programming language to Java.

Table 6.24: Kruskal-Wallis analysis results from the post-project survey question: *How would you categorize the difficulty of the project?*

Survey #	Mean rating			P-value
	.NET	CS1 Sandbox without subsets	CS1 Sandbox with subsets	
#1	2.1	2.3	2.4	0.31
#2	2.5	2.3	2.2	0.20
#3	2.5	2.5	2.4	0.71
#4	3.6	3.5	3.4	0.67
#5	3.0	3.0	3.1	0.91

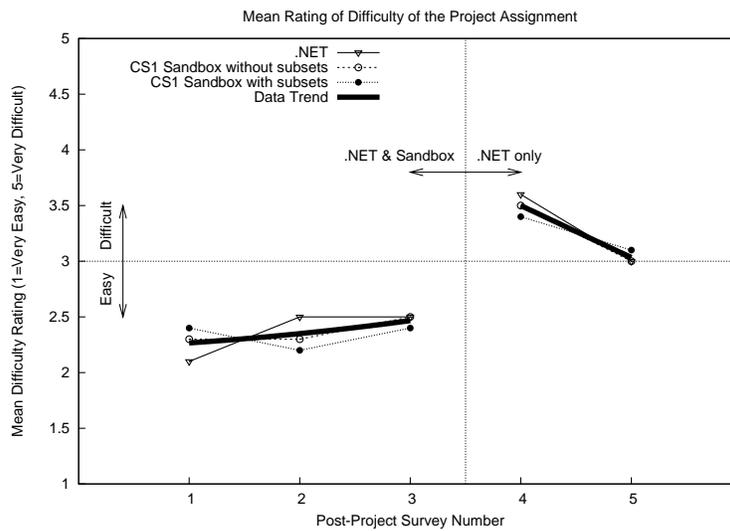


Figure 6.14: Graph of mean survey ratings of project difficulty with data trends

6.4.5.7 Post-Project Survey Question: Rating Frequency of Use of Built-In Help

Post-project survey question #10 attempted to obtain a measure of the amount of in-environment help used by the subjects while completing their programming projects. Kruskal-Wallis tests performed on the resulting data set (see Table 6.25) indicate that only the means from the first and third surveys had unequal means. The remaining three means (projects #2, #4, and #5) were found to be statistically equivalent (refer to Figure 6.15), as the P-values reported from the Kruskal-Wallis tests were 0.05, 0.12, and 0.11 respectively.

Table 6.25: Kruskal-Wallis analysis results from the post-project survey question: *How often did you use the in-environment help?*

Survey #	Mean rating			P-value
	.NET	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets	
#1	1.2	1.6	1.7	0.00
#2	1.2	1.4	1.2	0.05
#3	1.1	1.4	1.4	0.02
#4	1.2	1.5	1.5	0.12
#5	1.1	1.3	1.3	0.11

Further statistical analysis on the results from post-project surveys #1 and #3 was conducted to determine which of the experiment groups differ among these surveys. This analysis was achieved by executing pair-wise Kruskal-Wallis tests on each of the three pairs of possible combinations of the data. As reflected in Table 6.26, the .NET group once again differed from the *CS1 Sandbox* groups in both of the post-project surveys.

Table 6.26: Pair-wise Kruskal-Wallis analysis results from post-project surveys #1, and #3 – question #10

	Post-Project Survey P-values	
	#1	#3
.NET vs. <i>CS1 Sandbox</i> without	0.00	0.20
.NET vs. <i>CS1 Sandbox</i> with	0.00	0.20
<i>CS1 Sandbox</i> with vs. <i>CS1 Sandbox</i> without	0.63	0.94

Despite the acceptance that three of the five sets of means are equal in nature, it is noted by examining the graph of the means (see Figure 6.15) that experiment group with the lowest reported use of the in-environment help were the .NET subjects (consistently scoring 1.1 or 1.2) implying that they were less likely to use or require the use of in-environment help subsystems. In fact, none of the means are sufficiently high enough to conclude other than the built-in help was rarely used (if ever).

This observation comes as little surprise as very little of the course syllabus is dedicated to discussing the help available in the programming environment. The .NET programming environment provides a rather complete but complex help system covering far more material than would be needed by novice programmers. The *CS1 Sandbox* programming environment also contained a number of help files and a glossary of terms to assist in the comprehension of error messages, as well as possible likely causes and example implementations of program statements. As a result of this lack of use of the built-in help systems, this researcher speculates that additional focus in obtaining assistance from these features (on-line glossaries, help files, sample code, and error explanation guides) may be a useful addition to the course syllabus.

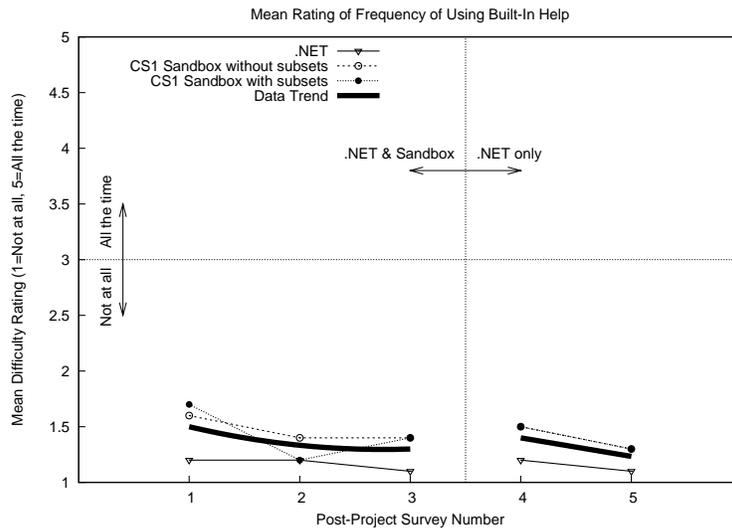


Figure 6.15: Graph of mean survey ratings of the frequency of use of built-in help with data trends

6.4.6 Observations from the Post-Project Survey Write-In Comments

Through the write-in comments portion of the post-project surveys (see Section 5.2.5.3), the users had the opportunity to provide invaluable feedback on their experiences with their programming environment (both positive and negative). As there are no statistical tests to perform on this data, a summarization of their comments are in order.

Users of the .NET programming environment (including those who migrated to .NET after using the *CS1 Sandbox*) had a number of positive comments about .NET’s debugger and syntax color highlighter. Of particular note were comments by individuals regarding a menu of function choices and the auto-completion of structure member variable names to help speed the code development process.

Particularly notable negative points raised in regards to the .NET programming environment include vague error messages, the slow and tedious startup process (both for starting the application and starting a new project), the “clutter” of the graphical user interface, and the “over-abundance” of features in .NET.

CS1 Sandbox users were not without their opinions on the *CS1 Sandbox* environment. Positive comments about the simplicity of the interface literally poured in, as well as compliments on the ease of application startup, the help system, and even the auto-update feature – providing them with the latest update to the environment (see Section 3.2.5).

Drawbacks to the *CS1 Sandbox* include poor error messages, anomalous interface issues (syntax color highlighting stopping without notice, the cursor disappearing, and the line and column indicator incorrectly calculating the position of the cursor). Serious concerns were expressed compiler errors that were caught by the Curator system⁸ but not flagged by the *CS1 Sandbox*.

Finally, the application of the language subsets themselves elicited just four comments from the subjects who encountered them. As there are only four and are germane to a major functionality issue in the prototype environment, they are highlighted again:

⁸The Curator executes a command line version of the .NET compiler to compile and execute student submissions.

“the subset feature was confusing...but i understand that it was necessary. Sometimes; however; I would do all my code and compile and i had forgotten about the subsets. Not a huge problem; but a little bit of an irritation”

“subsets; language not fully implemented”

“I don’t like having to download subsets; and there was a small issue with printing zeros.”

“I dislike the restraints set by the environment. For example; when one needs to use conditional statements; or absolute value functions the most; they are not enabled.”

From these comments this author concludes the following thoughts.

- Despite the best development efforts of the programming team for the *CS1 Sandbox*, graphical user interface issues which we considered minor in nature clearly bothered the subjects.
- Compiler consistency was one of the biggest concerns of the programming team, and for the most part few students reported issues in that department.
- Complexity matters to the students! Project management tools are not needed by the students in a CS1 course and many of them recognized that and have little desire to be forced to worry about such issues. The students do overcome the issue of a complex project start-up and will be better served by it later in their curriculum, but this author maintains that it is an issue that can be deferred until a later (more appropriate) time in their studies. The subjects responded rather positively with regards to the simplified interface of the *CS1 Sandbox*.
- Error messages still can be improved for novice programmers. Users of all of the programming environments in this study noted that error messages are too vague, too complex, or are unintelligible. The simple fact that so many subjects took the time to complain about error messages tells us that the messages are important to them – likely more so than for experienced programmers. We (as programmers and developers of compilers for novices) can and should do better.

6.4.7 Out-of-Laboratory Summary

The out-of-laboratory portion of the experiment certainly provided the bulk of the data analyzed by this dissertation. This included the electronic data automatically collected by use of the *CS1 Sandbox* programming environment (the number of compilation/execution attempts and the number of compilation errors), which in turn subsequently led to the derived data (time-on-task). Additionally, the project scores, and post-project surveys (self-estimated data, rating data, and write-in comments) contributed to the data sets examined.

Of the numerous statistical analyses performed on the data sets related to the out-of-laboratory projects, few individual results were significant. However, despite these results a number of trends and observations did emerge and are worth summarizing here again.

1. Project scores – a fairly consistent trend emerged despite variations in the complexity of the programming task, or in the case of the *CS1 Sandbox* programming environment problems with the programming environment itself.
2. Compilation/execution attempts – while using the *CS1 Sandbox*, users showed a trend of a decreasing number of compilation attempts despite the complexity of the program assignment increasing. A hypothesis from this result is that this is an indication of the users showing signs of increasing

acclimation toward the programming environment. Additionally, users of the *CS1 Sandbox* environment with subsets consistently attempted fewer compilations than their peers using the *CS1 Sandbox* without subsets.

3. Self-estimated number of compilations – users of the *CS1 Sandbox* (both with and without subsets) consistently underestimated the number of compilation/execution attempts they performed. One possible cause of this would be that the simplified environment helps to reduce the user’s perception of the amount of “work” they are performing.
4. Rating the difficulty of debugging code – users of the *CS1 Sandbox* (both with and without subsets) consistently rated their experience in debugging in the prototype environment as more difficult (but not significantly so). Combine this data with their written reactions (see Section 6.4.6) of their experiences completing the projects and we can certainly conclude their desire to utilize debugging tools (debuggers and/or variable watchers) despite not having been presented with instructions on how to effectively use them in completing their work.

6.5 Statistical Evaluation of Error Recognition

The data collected as the result of the error recognition experiment (a correct or incorrect mark for each possible error attempted to be recognized by each subject) was divided into four data sets – one for each type of error. These resulting sets (syntax, semantic, logic, and programs without errors) were each then statistically tested against the experiment groupings using the chi-squared method of analyzing nominal values to determine if the means were statistically equivalent.

It is hypothesized by this author that subjects who had used the *CS1 Sandbox* software (either version) would outperform (have a higher recognition rate of errors – based on the number of correctly answered questions) their peers who had been using the .NET programming environment. By analyzing the mean number of correct and incorrect answers for each group of experiment subjects, it can be determined if one or more of the groups were likely to outperform other groups of subjects.

6.5.1 Analysis of Error Recognition Experiment Results

Tables 6.27 through 6.30 show both the data collected from the error recognition experiment and the chi-squared results and corresponding P-values for the tests performed.

Table 6.27: Chi-Squared Test of Subject Groups vs. Recognition of Syntax Errors

	.NET	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets
# Correct	47	28	46
# Incorrect	8	7	9

Chi-squared result: $\chi^2 = 0.46$

Degrees of freedom: 2

P-value: 0.79

Table 6.28: Chi-Squared Test of Subject Groups vs. Recognition of Semantic Errors

	.NET	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets
# Correct	29	17	29
# Incorrect	15	11	15

Chi-squared result: $\chi^2 = 0.25$

Degrees of freedom: 2

P-value: 0.88

Table 6.29: Chi-Squared Test of Subject Groups vs. Recognition of Logic Errors

	.NET	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets
# Correct	7	2	7
# Incorrect	15	12	15

Chi-squared result: $\chi^2 = 1.63$

Degrees of freedom: 2

P-value: 0.44

Table 6.30: Chi-Squared Test of Subject Groups vs. Recognition of Source Code Without Errors

	.NET	<i>CS1 Sandbox</i> without subsets	<i>CS1 Sandbox</i> with subsets
# Correct	22	14	22
# Incorrect	11	7	11

Chi-squared result: $\chi^2 = 0.00$

Degrees of freedom: 2

P-value: 1.00

By performing the chi-squared tests on the each of the four data sets and analyzing the resulting P-values, it was found that in all cases the P-values were greater than the 0.05 significance level required to reject the null hypothesis (that the means of the groups were equal). Therefore, it is concluded that there is no statistical difference between the experiment groups when it comes to the recognition of syntactic, semantic, or logic errors. Additionally, no experiment group was more successful in recognizing program source code that contained no errors.

Beyond the statistics however, we find a disturbing trend. As Table 6.31 indicates, the subjects as an overall group did reasonably well in identifying syntax errors (85.8%). However their performance in the semantic and logic error recognition sections was considerably lower (64.7% and 27.6% respectively). This is likely a strong indication that the course syllabus is lacking in coverage of teaching how to recognize semantic and logic errors without actually authoring and executing source code (i.e. lack of source code walkthroughs/reviews).

Table 6.31: Summary of key error recognition experiment performance (experiment groups combined)

	% Correct
Recognition of syntax errors	85.8%
Recognition of semantic errors	64.7%
Recognition of logic errors	27.6%

6.5.2 Error Recognition Summary

Among all of the chi-squared tests executed on the data set collected from the error recognition experiment, none yielded significant statistical results which would indicate a result other than the mean rate of error identification to be equal among the experiment groups. These results seem to indicate that no programming environment used in this experiment contributed to improving the subjects' error recognition capabilities.

In a broader sense, this non-result is a positive. That is, if none of the programming environments had a negative effect on the identification of these types of common programming errors, then we can rule out the environment experimentation as having a negative effect on the subjects. This is clearly a victory for the development and experimentation of new programming environments – especially those which lack the more powerful features of the professional environments. In other words, despite the *CS1 Sandbox* lacking debugging tools, and in spite of problems identified (such as occasional issues with syntax color highlighting) by the subjects (see Section 5.2.5.3), the subjects using the experimental (*CS1 Sandbox*) programming environment were no worse off than their peers in recognizing potential errors in programs. Because of this result, this author argues that such work in experimental programming environments should not only continue, but be used in real course settings when testing the results of the experimental environments.

6.6 Observations from the Focus Group Transcript

The focus group transcript provided additional comments from the subjects about their likes and dislikes of their programming environment. The majority of these comments were very similar in nature to the

write-in comments from the post-project surveys (see Section 5.2.5.3) and provided confirmation of the veracity behind the write-in comments.

Many issues of programming environment functionality, or desired operations were elaborated on. These topics (for example, automatic file extension filtering and appending, auto formatting of source code, and cut-copy-paste behavior to name a few) are particularly useful for future work in developing programming environments for novices. Many of these topics were issues that the *CS1 Sandbox* development team felt the users would not need, so the elaboration of them did not go unnoticed.

A final observation should be mentioned about the subjects and their opinions. This researcher found the subjects participating in the focus group (and those that did provide write-in comments on the post-project surveys) were somewhat passionate about the software they were using. While many were happy to participate in the experiment and use a prototype environment, they also seemed to be willing and able to respond to opinions they expressed about the environment(s) they used during the experiment. It is fair to say that the programming environment is a key resource in the CS1 course, and that providing the students with a comfortable experience is a significant concern for the students.

Chapter 7

Conclusions, Contributions, Observations, and Future Work

7.1 Conclusions

Conclusions of this work are divided into the two main categories from the research questions presented earlier. These include conclusions on the impact of the use of the simplified environment and of the application of language subsets on the subjects' effort in completing their programming assignments (both closed-laboratory and out-of-laboratory).

7.1.1 Impact of the Use of a Simplified Environment

The subjects who used simplified programming environments had very positive responses. The conclusions drawn from the analysis of the data collected during the experiment come from the following areas:

- raw closed-laboratory and out-of-laboratory project scores,
- post-project surveys, and
- focus group comments.

These data are included in the analysis of the impact of the use of a simplified environment as they most closely relate to a student's perception of and reaction to such an environment's use.

While there were few significant statistical differences or consistent trends emerging from the raw scores (closed-laboratory and out-of-laboratory projects), these data (see Figures 6.1 and 6.4) indicate a positive conclusion for the use of a simplified environment on two fronts. First, subjects who used the simplified environment (and later migrated to the professional environment) performed equally as well as those that used the professional environment for the entire semester. That is, all of the students performed statistically equally for all of their programming course work, and thus it can be concluded that the application of the simplified environment in no way hinders the capabilities of the subject to achieve scores equal to their peers who use the conventional, full-featured programming environment.

Second, these results are the first of their kind that give any indication as to a quantitative impact of the use of a simplified environment for novice programmers. With users of the simplified environment scoring

as well as their peers using the more complex environment, this author argues that there is little reason not use a simplified environment in the early weeks of a introductory programming course. Certainly the length of use is likely highly dependent on many factors of each institution's subject's demographics, such as prior programming experience, coverage of topics, length of course, etc. As the result of this analysis, we now have an baseline indication of what impact a simplified environment is likely to have on its users (in terms of scores).

In addition to the closed-laboratory and out-of-laboratory project scores, the post-project surveys and focus group comments provided strong indicators that the *CS1 Sandbox* was preferred over .NET by the subjects that used both *CS1 Sandbox* environments. A very large number of subjects commented that they liked the *CS1 Sandbox* programming environment, commenting in line with one subject who said: *"I enjoyed the simplicity of Sandbox. While it may seem 'dumbed down' to some; it is perfect for beginners such as myself."* One subject who apparently had previously seen or worked with .NET included *"Very simple to use. Cuts a lot of the 'professional' features of .NET which aren't needed at the academic level; which makes it much simpler to focus on the code."*

From among the comments and survey data were clearly lacking any complaints by subjects who used both *CS1 Sandbox* environments that the migration process had any type of negative effect. In fact, as the subjects migrated, the post-project surveys for projects #4 and #5 the surveys indicated that the subjects found that it was more difficult to start a new project or program due to the professional environment's project management overhead. Comments from the surveys noted the displeasure of the lack of a program debugger in the *CS1 Sandbox*, however this was expected as the *CS1 Sandbox* was designed not to have one present.

This author concludes that despite a few environment bugs present in the *CS1 Sandbox* and the lack of the professional features, subjects clearly enjoyed working with the *CS1 Sandbox* and doing so presented no significant ill effects. With this baseline of data, the *CS1 Sandbox* programming environment can now be modified and improved upon. Such supplemental work would include the addition of a debugger, the removal of the remaining known software deficiencies (bugs) and completion of the grammar recognized by the *CS1 Sandbox* (i.e. arrays, enumerated types, structs, unions, etc.). Finally, the *CS1 Sandbox* can be extended as a platform for research in using programming environments to help improve semantic and logic error recognition performance (by the students).

7.1.2 Quantitative Impact of Subsets on User Effort

The subjects who used language subsets in their programming environment had very positive responses. The conclusions drawn from the analysis of the data collected during the experiment come from the following areas:

1. the mean number of compilation and execution attempts from the closed-laboratories and out-of-laboratory programming assignments,
2. the mean number of compilation and execution errors from the closed-laboratories and out-of-laboratory programming assignments, and
3. the mean time-on-task from the out-of-laboratory programming assignments.

These data are included in the analysis of the impact of the subsets on the effort required to use such an environment as they most closely relate to such measures of effort.

Two principal conclusions are drawn from the analysis of the impact of subsets on the subjects' effort. First, despite the absence of statistically significant differences in the mean number of compilation and

execution attempts (from both the closed-laboratories and out-of-laboratory programming assignments), there exists a clearly defined ordering of the experiment groups in the results. The results have shown (see Figures 6.2 and 6.6) that in every case where the students were not provided with a large percentage of the source code solution (as was the case in closed-laboratories #1 and #2), the users of the *CS1 Sandbox* with subsets had a few mean number of compilation and execution attempts than those users of the *CS1 Sandbox* without language subsets applied. Further analysis would be needed to determine the exact cause of these results, however with all other variables equal (the interface of the programming environment was the same, the underlying compiler was identical, the assignments were identical), clearly the application of the subsets had somewhat of a positive effect in terms of the effort required to complete the programming tasks.

Second, in terms of the mean number of compilation and execution errors (see Figures 6.3 and 6.7), and the mean time-on-task to complete the projects (see Figure 6.8), there was no significant statistical difference. These results permit this author to conclude that the application of language subsets had no negative effect on the effort expended by the subjects to complete their programming tasks. This is viewed alone as a significant result from this experiment as it is a clear indicator that the application of language subsets can likely be expanded upon with the confidence that undue effort will not be required of the subjects of future experiments.

7.2 Contributions of this Work

The work from this dissertation contributes to the greater field of computer science education in three principal ways.

1. This dissertation is one of the first evaluations of the quantitative effect of programming environments and language subsets. As detailed in Chapter 2, only a handful of the programming environments developed previously attempted a subjective analysis of their impact on students. None of these efforts undertook a quantitative analysis of the impact of the environments on students' effort (work) in completing programming assignments. The work of this author provides a benchmark for a further investigation of these areas.
2. This work showed that the application of subsets on the user interface of a programming environment and in the underlying language used by the students ultimately had no deleterious effect on the subjects. These results encourage the extension of this work without fear of ill effects upon future subjects.
3. Finally, this experiment showed that a simpler (less complex) programming environment does not hinder the later transition to a more complex (professional) program development environment by novice programmers.

7.3 Observations from this Work

A number of observations also emerged in two areas from this work. These include implications on program development environments themselves and implications on the development of CS1 syllabi. These are discussed in the sections that follow.

7.3.1 Implications on Programming Environments

Several strong opinions and thoughts emerged from the experiment's subjects with regard to the environments with which they worked. In nearly every case, their comments include content which should be strongly considered in future endeavors in programming environments for novice programmers. The most prevalent include:

1. Error messages – One clear message from the subjects (regardless of the environment they used) in this experiment is the need for clearer error messages. The *CS1 Sandbox* environment presented several messages which the subjects felt were not helpful at all and may have hindered the completion of the assignment (by increasing the time-on-task) even though special development attention was applied to this topic. Not unexpectedly, users of the .NET programming environment commented on the confusing messages – principally designed for experienced programmers.

Compiler messages are usually the first indication¹ that there exists serious problems with a student's source code. Error messages (include those of the *CS1 Sandbox*) need to be made even more clear and helpful to novice students. This is a rather difficult task as a compiler's internals (lexer, parser, etc.) may have a great deal of difficulty in discerning what may be the intention of the entered strings. This was certainly the case for the *CS1 Sandbox*, as despite our best efforts to present useful error messages, as suggested by the student developers, subjects still found many of them unhelpful.

From these observations, it seems straightforward to conclude that error messages continue to require attention from developers and instructors alike. Perhaps efforts should be directed toward the compiler community to have the production of "novice-level" error messages be an option at compile time. Alternatively, program development environments (which may rely on an external compiler) could filter error messages and essentially rewrite them prior to presentation to the user.

2. Start up / Project Management – Project initialization also is clearly a hot topic for novice users, no matter the environment used. *CS1 Sandbox* users (following their migration to .NET) commented strongly negatively via their rating of the startup difficulty and via the write-in comments from the post-project surveys about .NET's complexity in getting to the point where the subjects could just enter source code and proceed to complete their assignments. This researcher therefore suggests the use of simpler program/project initialization and project management sequences.

Common sense seems to indicate that at the initial stages of the CS1 course, when the most simple of assignments are being distributed, this is the time when interface and environment complexity needs to be avoided the most. For the most part, CS1 courses are taught without delving into the topic of multiple source code files for a single executable (when C or C++ is the language of choice). Applying the Java programming language to the CS1 curriculum has made this a semi-moot point as Java requires the use of a source code file for each public class. In today's language domain, without going into great pains at the introductory level, it is difficult to avoid discussing multiple source code files.

BlueJ's interface (see Section 2.3.4) masks this situation reasonably by forcing the user to operate with rectangles which represent classes the student is developing. File manipulation has been successfully abstracted away in the BlueJ programming environment. This is a positive gain in the reduction of complexity of the environment and the interface, yet may lead to lack of understanding on the part of students regarding how compiled programs are truly assembled².

¹Syntax color highlighting may in fact give clues to a program's correctness prior to any compilation attempt. Therefore, its contribution as feedback to the student programmer should not be discounted.

²This author experienced similar confusion from a student previously who had only worked in the Visual C++ environment. He exhibited no perception of how to command line compile a program or what role the linker played due to the heavy abstraction of these concepts throughout his curriculum.

3. User interface – Here again, .NET subjects mentioned the complexity of the interface and/or the number of tools and options being overly taxing in working in the programming environment. These concerns were echoed (even more loudly) by *CS1 Sandbox* users following their migration to the .NET environment.

Complex environments are fine for professionals, or even students who have a semester or two of programming under their belt. However, this author argues that we are performing a disservice to the novices in the community by having them use a professional strength environment initially. This potential over-assumption of the ability of a novice student to overcome the complexities of the environment may in fact work against the student by increasing their anxiety of programming or program development.

This researcher concludes that it seems that a balance needs to be struck between the complexity of the interface and the support provided to organize a user's source code into a manageable project. This experiment has clearly indicated that the professional programming environment currently used does not meet these goals and a simplified one (such as the *CS1 Sandbox*) comes closer to the needs and desires of the student programmers.

7.3.2 Implications on the Syllabus of CS1 Courses

Of particular note from this work are several implications on the syllabus of CS1 courses. These include the following:

1. Built-in Help – Post-project survey results (see Figure 6.15) seem to indicate that students rarely use the built-in help system present in programming environments. Those that do, seemed to appreciate efforts to help present explanations of error messages and sample code at a level which helps facilitate a clearer understanding of the issues confronting them while compiling. From these observations, it would seem that students are either unwilling to use built-in help (and prefer other sources of help such as a teaching assistant or their course textbook/notes), or they do not know it exists.

This author concludes that in the syllabi of CS1 courses an additional measure of coverage should be given to utilizing built-in help and therefore providing additional instruction on how to effectively use the programming environments. All too often students (certainly the case here at Virginia Tech) may be left to fend for themselves to learn how to use results seem to indicate that students rarely use the built-in help system present in programming environments. This is especially apropos when utilizing a professional strength programming environment in conjunction with an introductory course.

Leaving students to self-discover how to utilize the tools they are expected to use in completing their assignments often (as observed by this researcher as a course instructor and teaching assistant in the CS1 course) leads to increasing the frustration level of novice students.

Novice programming students have a tough row to hoe. As mentioned in Chapter 1, there are three areas which they are expected to master:

- (a) learning the language of initiating a project and of an editor (see Section 1.1.1),
- (b) learning the language of the programming paradigm (see Section 1.1.2), and
- (c) learning the language of the programming environment (see Section 1.1.3).

Failing to provide adequate coverage of effectively using the tools of the trade at even the introductory level is a disservice to those students who do not arrive at the university with prior programming experiences.

2. Additional coverage of semantics and logic – The data presented from the Error Recognition Experiment is, simply put, shocking. The ability of the subjects (or lack there of) to identify a variety of semantic and logic errors clearly indicates that they are lacking knowledge and experience in this realm.

Virginia Tech’s Association for Computing Machinery (ACM) Programming Team is widely known for its success. This is largely attributable to how they train – each week studying problem sets and solution strategies. In addition to training in the likely categories of problems they will face, they are also heavily exposed to being able to correctly and quickly, develop and test a solution in the same manner as they will find themselves during the contest – away from a machine and coding on a piece of paper. Each programming team, during the contest, is permitted to use only one computer to enter and submit their solutions. While one programmer is entering the solution for one problem, other team members are working out solutions by hand, and on paper, elsewhere. Their performance clearly indicates that students can be trained to recognize their semantic and logic errors, and can be successful in doing so. A programming team member does not waste time debugging logic errors when (s)he finally gets to the keyboard during the contest.

This researcher concludes that these two programming areas need to be bolstered within the syllabus of the CS1 course. Clearly it would seem that the CS1 course at Virginia Tech is doing a relatively fine job of teaching the syntax of a programming language. However, the poor results from the semantic and logic portions of the recognition experiment flag a dangerous trend that our students apparently have little experience or success in the broader points (identification of the more subtle errors) of programming. A potential research question for this work would be how an improved *CS1 Sandbox* can help with these recognition deficiencies.

7.4 Future Work

Given that the *CS1 Sandbox* programming environment was a prototype application designed and developed specifically to test the research questions presented in this dissertation, several areas of additional work have emerged for consideration.

1. Further examination of iterative/incremental development – With over 45,000 database records of subject programming submissions resulting from this study, a large pool of iterative and incremental program development exists. This researcher, while perusing the database, noted that one can extract all the submissions for a student (or students) for a particular assignment. From this data, abstract syntax trees could be formed and become the basis for an examination of how students develop their programs by examining how the AST trees develop. This author hypothesizes that AST tree/program development likely varies widely from student to student, but may yield interesting trends based on a subject’s prior programming experience.
2. More detailed examination of time-on-task – As noted above, a large amount of programming data was collected by this experiment. In addition to the examination of the ASTs generated as a subject programs, an examination of *how* students complete their programming activities could be undertaken by examining the data relative to the timestamps contained in each record. Such an examination could yield interesting insight as to how long students program for, the length of time students spend debugging, and how long students perform testing. Additionally, this data can also elucidate correctly when students start programming (early? late? last minute?) once given a program specification.
3. Repeat using the Java programming language, use a greater percentage of novice programmers – Since the initial development of the *CS1 Sandbox*, the Java programming language has become the de

facto standard programming language in CS1 courses across the globe. Continued exploration of the subset concept will likely require the migration of the programming environment to support the Java language, especially as the proliferation continues (generally away from C/C++).

A likely course of action would be to add data collection and subset modules to the BlueJ [41] (see Section 2.3.4) programming environment. Recent versions of BlueJ have begun to support a framework for extending the programming environment via a custom application programming interface (API). By capitalizing on this framework, the subset idea can likely be revisited in Java by one of today's most popular programming environments designed for novice users.

Finally, any re-examination of this experiment should likely include a larger pool of subjects with no prior programming experience. Of the 165 subjects in this study, only 7 (4.0%) reported no prior programming experience. Additionally nearly a quarter (23.6%) rated their prior experience from slight to good in one or more languages (see Appendix B). Clearly, the typical computer science major that enrolls at Virginia Tech comes to campus with some measure of prior knowledge. A more thorough examination could be conducted involving true novices with no prior experience. However this should likely not occur at Virginia Tech.

4. Investigate the impact of new features – The single most notable new feature that this author would add to future versions of the *CS1 Sandbox* would that of a debugger. Despite the lack of significant coverage of the effective use of a debugger in the course syllabus, subjects who used the .NET programming environment specifically noted its effectiveness. Additionally, some subjects who had used the *CS1 Sandbox* environment specifically noted the absence of a debugger. Recognition of the use of a debugger may be a symptom of these subjects having prior programming experience, or it may come from the sharing of environment hints and tips with their peers. Regardless, this author concludes that some fundamental program debugger (one that supports watching values associated with variables and step-wise execution of a program) should be added as a feature.
5. Staged deployment of the PDE's interface – Similar to the language subsets implemented in the *CS1 Sandbox*, the notion of subsets can be added to the user interface of the programming environment. In doing so, instructors would have the ability to grow the user interface's complexity and environment's capabilities as the students become acclimated with the programming task.
Such deployment would likely include features such as syntax color highlighting, a debugger, project management tools, or even the ability to prettyprint the source code for the student. By providing the ability to subset the environment's features and user interface components, students can use a programming environment which grows with them – first concentrating on achieving success with initial programming activities (source code file creation, successful compilation and execution) using a very simplified environment, and moving in time to a more complex and capable programming environment which provides a richer set of tools.
6. Slow down the ability to use/access the program development environment to force critical thinking and thereby reduce the number of compilations – This researcher noted the large number of compilation attempts in both the closed-laboratory and out-of-laboratory programming assignments. In some cases, students appeared (based on examination of the number of compilations and the time between compilations) to be solely attempting re-compilation of source code in the hopes that syntax and semantic errors would simply disappear. To some degree this is perceived as a measure of frustration with a programming environment. However, an alternative point of view on such behavior is that the students are quickly making a small change and relying on the compilation attempt to validate their rapid guess at the solution to an error. That is, such students do not seem to be thinking critically about the problems they are encountering.

It is hypothesized that if the program development environment were structured to prevent such rapid compilation attempts (perhaps via a timing mechanism) students would more carefully examine

the error(s) presented to them, and attempt a more thoughtful and thorough analysis of the problems at hand. Thus, a future direction of this work would be to extend the *CS1 Sandbox* to support such critical thinking behavior and to measure if such a design is not only practical but results in the intended results.

The *CS1 Sandbox* programming environment has proved to be a successful prototype platform for the exploration of a variety of ideas targeted toward the novice programmer. It is likely that many of the notions explored by this dissertation may well be re-evaluated again using different programming languages and with other programming environments.

Bibliography

- [1] ACT, INC. The American College Test (ACT). Internet WWW page, at URL: <http://www.act.org/> (last accessed 5/31/2003).
- [2] ALLEN, E., CARTWRIGHT, R., AND REIS, C. Production programming in the classroom. In *Proceedings of the 34th Technical Symposium on Computer Science Education* (New York, New York, 2003), Association for Computing Machinery, pp. 89–93.
- [3] ALLEN, E., CARTWRIGHT, R., AND STOLER, B. DrJava: A lightweight pedagogic environment for Java. In *Proceedings of the Thirty Third SIGCSE Technical Symposium on Computer Science Education* (New York, New York, February 2002), Association for Computing Machinery, pp. 137–141.
- [4] APPLEBY, D., AND VANDEKOPPLE, J. J. *Programming Languages: Paradigm and Practice*. McGraw-Hill Companies, Inc., 1997, ch. 0, p. 23.
- [5] ARON, A., AND ARON, E. N. *Statistics for the Behavioral and Social Sciences*. Prentice Hall, Englewood Cliffs, New Jersey, 2001, ch. 10, p. 200.
- [6] AUSTING, R. H., BARNES, B. H., BONNETTE, D. T., ENGEL, G. L., AND STOKES, G. Curriculum '78: recommendations for the undergraduate program in computer science? a report of the acm curriculum committee on computer science. *Communications of the ACM* 22, 3 (1979), 147–166.
- [7] BARNES, D. J., AND KÖLLING, M. *Objects First with Java - A Practical Introduction using BlueJ*, 1st ed. Prentice-Hall, Inc. / Pearson Education, Englewood Cliffs, New Jersey, 2003.
- [8] BECKER, B. W. Teaching CS1 with Karel the Robot in Java. In *Proceedings of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education* (2001), ACM Press, pp. 50–54.
- [9] BERGIN, J., STEHLIK, M., ROBERTS, J., AND PATTIS, R. Karel J. Robot - A Gentle Introduction to the Art of Programming. Internet WWW page, at URL: <http://csis.pace.edu/~bergin/KarelJava2ed/Karel++JavaEdition.html> (last accessed 5/1/2003).
- [10] BERGIN, J., STEHLIK, M., ROBERTS, J., AND PATTIS, R. *Karel++ - A Gentle Introduction to the Art of Programming*, 1st ed. John Wiley & Sons, New York, New York, 1997.
- [11] BJORK, R. C. Objects-first. Posted to the ACM SIGCSE Listserv (SIGCSE.MEMBERS@ACM.ORG), March 2003.
- [12] BOARD, T. C. The scholastic aptitude test. Internet WWW page, at URL: <http://www.collegeboard.com/> (last accessed 5/31/2003).
- [13] BROOKS JR., F. P. No Silver Bullet: Essence and Accidents of Software Engineering. In *Information Processing 86* (North-Holland, September 1986), Elsevier Science Publishers B.V., pp. 1069–1076.

- [14] BUCK, D., AND STUCKI, D. J. JKarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum. In *Proceedings of the Twenty Seventh SIGCSE Technical Symposium on Computer Science Education* (2001), ACM Press, pp. 16–20.
- [15] CARROL, J. M., AND ROSSON, M. B. Deliberated Evolution: Stalking the View Matcher in Design Space. *Human-Computer Interaction* 6, 3 & 4 (1991), 281–318.
- [16] CONWAY, M., AUDIA, S., BURNETTE, T., COSGROVE, D., AND CHRISTIANSEN, K. Alice: Lessons Learned from Building a 3D System for Novices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, New York, 2000), Association for Computing Machinery, pp. 486–493.
- [17] COOPER, S., DANN, W., AND PAUSCH, R. Alice: a 3-D Tool for Introductory Programming Concepts. In *Proceedings of the Fifth Annual CCSC Northeastern Conference on The Journal of Computing in Small Colleges* (New York, New York, 2000), Association for Computing Machinery, pp. 107–116.
- [18] CROSS, J. H. GRASP - Graphical Representation of Algorithms, Structures and Processes. Internet WWW page, at URL: <<http://www.eng.auburn.edu/department/cse/research/grasp/>> (last accessed 4/3/2002).
- [19] CROSS, J. H. GRASP/jGRASP. Internet WWW page, at URL: <<http://www.eng.auburn.edu/grasp/>> (last accessed 2/27/2003).
- [20] CROSS, J. H. Introduction to the Complexity Profile Graph. Internet WWW page, at URL: <http://www.eng.auburn.edu/grasp/documentation/intro_to_CPG/index.html> (last accessed 2/27/2003).
- [21] CROSS, J. H. Introduction to the Control Structure Diagram. Internet WWW page, at URL: <http://www.eng.auburn.edu/grasp/documentation/intro_to_CSD/c_index.html> (last accessed 2/27/2003).
- [22] DEEK, F. P., AND MCHUGH, J. A. A review and analysis of tools for learning programming. In *Proceedings of the Tenth World Conference on Educational Multimedia and Hypermedia and on Educational Telecommunications* (1998), pp. 251–256.
- [23] DERAADT, M., WATSON, R., AND TOLEMAN, M. Language trends in introductory programming courses. In *The Proceedings of Informing Science and Information Technology Education Conference* (June 2002), Informing Science, pp. 329–337.
- [24] DIETEL, H. M., AND DIETEL, P. J. *C++ How To Program*, 3rd ed. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 2001, ch. 2, p. 63.
- [25] EISENSTADT, M. A User-Friendly Software Environment For The Novice Programmer. *Communications of the ACM* 26, 12 (December 1983), 1058–1064.
- [26] FELDMAN, M. B., LOPES, A. V., AND PÉREZ-QUIÑONES, M. SmallAda: Personal Computer Courseware for Studying Concurrent Programming. In *Proceedings of the Twenty-First SIGCSE Technical Symposium on Computer Science Education* (1990), ACM Press, pp. 206–211.
- [27] FELLEISEN, M., FINDLER, R. B., FLATT, M., AND KRISHNAMURTHI, S. Functional programming: The DrScheme Project: An overview. *ACM SIGPLAN Notices* 33, 6 (June 1998), 17–23.
- [28] FELLEISEN, M., FINDLER, R. B., FLATT, M., AND KRISHNAMURTHI, S. *How to Design Programs: An Introduction to Programming and Computing*. The MIT Press, Cambridge, Massachusetts, 2001.

- [29] FINDLER, R. B., FLANAGAN, C., FLATT, M., KRISHNAMURTHI, S., AND FELLEISEN, M. DrScheme: A Pedagogic Programming Environment for Scheme. *Programming Languages: Implementations, Logics, and Programs 1292* (September 1997), 369–388.
- [30] FREE SOFTWARE FOUNDATION, INC. The Concurrent Versioning System (CVS). Internet WWW page, at URL: <<http://www.gnu.org/software/cvs/cvs.html>> (last accessed 1/4/2003).
- [31] GAMMA, E., AND BECK, K. Junit Regression Testing Framework. Internet WWW page, at URL: <<http://www.junit.org/>> (last accessed 2/17/2003).
- [32] GOLD, E., AND ROSSON, M. B. Portia: An Instance-Centered Environment for Smalltalk. In *Conference proceedings on Object-oriented Programming Systems, Languages, and Applications* (1991), ACM Press, pp. 62–74.
- [33] GOSLING, J., AND ARNOLD, K. *The Java Programming Language*, 1st ed. Addison-Wesley, Reading, Massachusetts, 1996.
- [34] GRIES, D., AND GRIES, P. ProgramLive: A Multimedia, Java-based Livetext on Programming. Internet WWW page, at URL: <<http://www.datadesk.com/ProgramLive/>> (last accessed 5/1/2001).
- [35] GUZDIAL, M. The Role of Artifacts in Programming and Physics Learning with Emile, April 1995.
- [36] GUZDIAL, M. Software-realized scaffolding to facilitate programming for science learning. *Interactive Learning Environments 1*, 4 (1995), 1–44.
- [37] HOLEY, J. A., AND ZIEGLER, L. R. Introduction to the Science of Computing. Unpublished manuscript, reviewed Spring of 2001 by John A.N. Lee, 2000.
- [38] HOLT SOFTWARE ASSOCIATES, INC. Ready to Program. Internet WWW page, at URL: <<http://www.holtsoft.com/ready/>> (last accessed 6/27/2003).
- [39] HUME, J. P., AND STEPHENSON, C. *Introduction To Programming In Java*. Holt Software Associates, Inc., 2000.
- [40] KÖLLING, M. Why BlueJ? An introduction to the problems BlueJ addresses. Internet WWW page, at URL: <<http://bluej.monash.edu/why/why.html>> (last accessed 2/5/2003).
- [41] KÖLLING, M., AND ROSENBERG, J. BlueJ - the Interactive Java Environment. Internet WWW page, at URL: <<http://bluej.monash.edu>> (last accessed 2/5/2003).
- [42] LANE, D. M. HyperStat Online Textbook – Significance Level. Internet WWW page, at URL: <<http://davidmlane.com/hyperstat/A72117.html>> (last accessed 7/1/2003).
- [43] LEE, J. A. N. Private conversation re: Ada, June 2003.
- [44] LEE, J. A. N. Private conversation re: Smalltalk, June 2003.
- [45] LUKAS, G. Uses of the LOGO Programming Language in Undergraduate Instruction. In *Proceedings of the ACM Annual Conference* (New York, New York, 1972), Association for Computing Machinery, pp. 1130–1136.
- [46] MACLENNAN, B. J. *Principals of Programming Languages: Design, Evaluation, and Implementation*. Oxford Univeristy Press, 1999, ch. 2, p. 291.
- [47] MACLENNAN, B. J. *Principals of Programming Languages: Design, Evaluation, and Implementation*. Oxford Univeristy Press, 1999, ch. 7, p. 246.

- [48] MANDRAKESOFT SA. Mandrake Operating System. Internet WWW page, at URL: <<http://www.mandrakelinux.com/en>> (last accessed 5/11/2003).
- [49] MCQUAIN, W. Student Guide to the Curator. Internet WWW page, at URL: <<http://www.cs.vt.edu/curator/StudentInfo/CuratorStudentGuide.doc>> (last accessed 6/13/2003).
- [50] MCQUAIN, W., AND ET. AL. Curator: an Electronic Submissions Management Environment. Internet WWW page, at URL: <<http://www.cs.vt.edu/curator>> (last accessed 1/4/2003).
- [51] MICROSOFT CORPORATION. Common Type System. Internet WWW page, at URL: <<http://msdn.microsoft.com/library/>> (last accessed 6/15/2003).
- [52] MICROSOFT CORPORATION. Microsoft Visual C++ .NET Standard. Internet WWW page, at URL: <<http://www.microsoft.com/catalog/display.asp?subid=22&site=11131>> (last accessed 6/15/2003).
- [53] MICROSOFT CORPORATION. Microsoft Visual C++, version 6.0. <<http://msdn.microsoft.com/visualc/default.asp>> (last accessed 5/1/2001).
- [54] MICROSOFT CORPORATION. What is the Common Language Specification? Internet WWW page, at URL: <<http://msdn.microsoft.com/library/>> (last accessed 6/15/2003).
- [55] MYSQL AB. The MySQL database server. Internet WWW page, at URL: <<http://www.mysql.com>> (last accessed 5/11/2003).
- [56] NEWBURGER, E. Home Computers and Internet Use in the United States: August 2000. Special Study P23-207, United States Department of Commerce, August 2000.
- [57] OBJECT MANAGEMENT GROUP. Unified Modeling Language. Internet WWW page, at URL: <<http://www.uml.org/>> (last accessed 2/8/2003).
- [58] OSTERMILLER, S. Syntax color highlighter. Internet WWW page, at URL: <<http://www.ostermiller.org/syntax/>> (last accessed 1/28/2003).
- [59] PAPERT, S. *Mindstorms: Children, Computers, and Powerful Ideas*, 1st ed. Basic Books, Inc., New York, New York, 1980.
- [60] PARR, T. ANTLR. Internet WWW page, at URL: <<http://www.antlr.org>> (last accessed 4/3/2002).
- [61] PATTIS, R. E., ROBERTS, J., AND STEHLIK, M. *Karel the Robot - A Gentle Introduction to the Art of Programming*, 2nd ed. John Wiley & Sons, New York, New York, 1995.
- [62] RAPID DEPLOYMENT SOFTWARE INC. The Official Euphoria Programming Page. Internet WWW page, at URL: <<http://www.rapideuphoria.com/>> (last accessed 4/4/2003).
- [63] ROBERTS, E. An Overview of MiniJava. In *Proceedings of the Twenty Seventh SIGCSE Technical Symposium on Computer Science Education* (2001), ACM Press, pp. 1-5.
- [64] ROBERTS, E., ET AL. Computing Curricula 2001. Internet WWW page, at URL: <<http://www.computer.org/education/cc2001/final/index.htm>> (last accessed 5/2/2001).
- [65] ROSSON, M. B., AND CARROL, J. M. Climbing the Smalltalk Mountain. *ACM SIGCHI Bulletin* 21, 3 (1990), 76.

- [66] ROSSON, M. B., CARROL, J. M., AND BELLAMY, R. K. E. Smalltalk Scaffolding: A Case Study of Minimalist Instruction. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (1990), ACM Press, pp. 423–430.
- [67] ROSSON, M. B., CARROL, J. M., AND SWEENEY, C. A View Matcher for Reusing Smalltalk Classes. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (1991), ACM Press, pp. 277–283.
- [68] RR SOFTWARE, INC. Janus/Ada 83. Internet WWW page, at URL: <http://www.rrsoftware.com/html/prodinf/janus83/j-ada83.htm> (last accessed 6/27/2003).
- [69] SAMMET, J. *Programming Languages: History and Fundamentals*, 1st ed. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1969.
- [70] SCHULMAN, R. Private correspondence, February 2003.
- [71] SCHULMAN, R. S. *Statistics in Plain English with Computer Applications*. Van Nostrand Reinhold, 1992, ch. 5, p. 166.
- [72] SEBESTA, R. W. *Concepts of Programming Languages*, 4th ed. Addison-Wesley, 1999, ch. 1, p. 9.
- [73] SLEEMAN, D. The Challenges of Teaching Computer Programming. *Communications of the ACM* 29, 9 (September 1986), 840–841.
- [74] STOLER, B. A Framework for Building Pedagogic Java Programming Environments. Master’s thesis, Rice University, April 2002.
- [75] SUN MICROSYSTEMS, INC. The Java Programming Language. Internet WWW page, at URL: <http://java.sun.com> (last accessed 1/18/2003).
- [76] SUN MICROSYSTEMS, INC. Java web start. Internet WWW page, at URL: <http://java.sun.com/products/javawebstart/> (last accessed 1/18/2003).
- [77] TEITELBAUM, T., AND REPS, T. The Cornell Program Synthesizer: A Syntax-Directed Programming Environment. *Communications of the ACM* 24, 9 (1981), 563–573.
- [78] THE APACHE SOFTWARE FOUNDATION. PHP: Hypertext Preprocessor. Internet WWW page, at URL: <http://www.php.net> (last accessed 5/11/2003).
- [79] THE APACHE SOFTWARE FOUNDATION. The Apache HTTP Server Project. Internet WWW page, at URL: <http://httpd.apache.org> (last accessed 5/11/2003).
- [80] TURINGSCRAFT, INC. Codelabs. Internet WWW page, at URL: <http://www.turingscraft.com/codelabs.php> (last accessed 2/26/2003).
- [81] VARIOUS AUTHORS. Introduction to Programming in C. Internet WWW page, at URL: <http://courses.cs.vt.edu/~cs1044> (last accessed 1/4/2003).
- [82] VARIOUS AUTHORS. ISO/IEC 9899:1999 Programming Languages – C. Internet WWW page, at URL: <http://www.iso.ch/cate/d29237.html> (last accessed 5/1/2001).
- [83] VIRGINIA TECH RESEARCH DIVISION. The Institutional Review Board for Projects Involving Human Subjects. Internet WWW page, at URL: <http://www.irb.vt.edu/> (last accessed 6/9/2003).
- [84] VIRGINIA TECH WEB APPLICATION RESEARCH AND DEVELOPMENT DEPARTMENT. survey.vt.edu. Internet WWW page, at URL: <http://survey.vt.edu/index.html> (last accessed 6/19/2003).

- [85] WEBBER, A. B. The Pascal Trainer. In *Proceedings of the Twenty Seventh SIGCSE Technical Symposium on Computer Science Education* (New York, New York, March 1996), Association for Computing Machinery, pp. 261–265.
- [86] WEXELBLAT, R. L. *History of Programming Languages*, 1st ed. Academic Press, 1981, ch. 5, pp. 199–277.
- [87] WEXELBLAT, R. L. *History of Programming Languages*, 1st ed. Academic Press, 1981, ch. 11, pp. 515–549.
- [88] WHITEAKER, W. A. Ada – The Project: The DoD High Order Language Working Group. In *History of Programming Languages – II* (1993), ACM Press, pp. 173–232.
- [89] WILKES, M. *Memoirs of a Computer Pioneer*. The MIT Press, 1985, ch. 14, p. 145.
- [90] WIRTH, N. *Systematic Programming: An Introduction*. Series in Automatic Computation. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973, ch. Preface, p. xiii.
- [91] WORLD WIDE WEB CONSORTIUM. HyperText Markup Language (HTML) Home Page. Internet WWW page, at URL: <http://www.w3.org/MarkUp/> (last accessed 6/21/2003).

Appendix A

IRB Application and Student Agreement

Proposal for
Research Study of
The Effect of a Simplified Programming Environment and
Programming Language Subsets on
Novice Computer Programmers

Submitted by
Peter J. DePasquale, Ph.D. Candidate
and
John A. N. Lee, Professor

Department of Computer Science
Virginia Polytechnic Institute and State University

Request for Exemption of Research Involving Human Subjects

Investigator(s): Peter J. DePasquale and John A.N. Lee

Department(s): Computer Science Mail Code: 0106 Email: pjdepasq@vt.edu, janlee@cs.vt.edu

Project Title: The Effect of a Simplified Programming Environment and Programming Language Subsets on Novice Computer Programmers

Source of Funding Support: Departmental Research Sponsored Research (OSP No.: _____)

All investigators of this project are qualified through completion of the formal training program or videotape program provided by the Virginia Tech Office of Research Compliance.

Note: To qualify for Exemption, the research must be (a) of minimal risk to the subjects, (b) must not involve any of the special classes of subjects, and (c) must be in one or more of the following categories. A full description of these categories may be found in the Exempt Research section of the instructions: "Application for Approval of Research Involving Human Subjects" or in the federal regulations [45 CFR 46.101(b)(1-6)]. (<http://grants.nih.gov/grants/oprr/humansubjects/45cfr46.htm#46.101>)

Please mark/check the appropriate category or categories below which qualify the proposed project for exemption:

- 1. Research will be conducted in established or commonly accepted educational settings, involving normal educational practices [see item (1), page ____].
- 2. Research will involve the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures or observation of public behavior, **unless** the subjects can be identified directly or through identifiers linked to the subjects **and** disclosure of responses could reasonably place the subjects at risk or criminal or civil liability or be damaging to the subjects' financial standing, employability or reputation [see item (2), page ____].
- 3. Research will involve the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures, or observation of public behavior that is not exempt under item 2) above **if** the subjects are elected or appointed public officials or candidates for public office; **or** Federal statute(s) require(s) that the confidentiality or other personally identifiable information will be maintained [see item (3), page ____].
- 4. Research will involve the collection or study of existing data, documents, records, pathological specimens, or diagnostic specimens if these sources are publicly available or if the information is recorded by the investigator in such a manner that subjects cannot be identified directly or through identifiers linked to the subjects [see item (4), page ____].
- 5. Research and demonstration projects designed to study, evaluate, or otherwise examine public benefit or service programs, procedures for obtaining benefits or proposed changes in such programs [see item (5), page ____].
- 6. Taste and food quality evaluation and consumer acceptance studies [see item (6), page ____].

	<u>Peter J. DePasquale</u>			
Investigator(s)	Print name			Date
	<u>John A. N. Lee</u>			
Investigator(s)	Print name			Date
	<u>Dennis Kafura</u>			
Departmental Reviewer	Print name			Date
Chair, Institutional Review Board				Date

Justification of Project

This study is an examination of the effect of the application of both a simplified programming environment and the introduction of programming language subsets on novice students in a first year programming course. The project detailed in this proposal seeks to collect data to determine the following:

- Does a simplified programming environment (not designed for professional developers, less graphical user interface components, less complexity) yield quantifiable changes in student performance in a novice programming course?
- Does the application of programming language subsets yield quantifiable changes in student performance in a novice programming course?

Data collected by the proposed project will be used by Peter DePasquale in support of his Ph.D. dissertation in Computer Science.

The project will be executed throughout the Fall 2002 semester, encompassing the roughly 240 Computer Science majors enrolled in the *CS1044: Introduction to Programming in C* course. The sections involved in this project are slated to be taught by Bill McQuain (Instructor, Department of Computer Science), and assisted by Peter DePasquale (Ph.D. candidate and teaching assistant, Department of Computer Science). Mr. McQuain has been consulted in the design and implementation of the requirements of this study as related to the course administration.

Procedures

Selection of students: The students participating in this investigation will be drawn from the CS1044 course offerings for Computer Science majors this Fall (2002). Currently there are two lecture sections offered, and it is anticipated that roughly 240 students will be enrolled in the course.

For this experiment we will draw our subjects entirely from the two sections. Approximately half of the students will be using the Microsoft Visual C++ 6.0 programming environment (or Visual C++ for .NET programming environment) as would normally be the case in the administration of CS1044. The remaining subjects will use one of two versions of the *CSI Sandbox*¹ programming environment (one with language subsets and the other without subsets), developed for this study.

Composition of project groups: Each student will be placed in one of three categories for the purposes of this investigation. The categories serve to dictate which programming environment the student will use through the first 60% of the course. The three environments being used are:

- Microsoft Visual C++ version 6.0² – Visual C++ is the standard programming environment the students are required to learn in CS1044.
- *CSI Sandbox* (without subsets) – Students in this group will utilize the *CSI Sandbox* programming environment without the use of language subsets (full language capabilities enabled).
- *CSI Sandbox* (with subsets) – Students in this group will utilize the *CSI Sandbox* programming environment and apply pre-configured language subsets to their environment. The subsets will be created by Peter DePasquale and tailored to correspond to the necessary language components

¹ *CSI Sandbox* is a programming environment created by Peter DePasquale in support of his dissertation. The environment has been developed and tested specifically for use by/with novice programmers in CS1044. Additional information about the software can be found at <http://sandbox.cs.vt.edu/~sandbox>

² At the time of the writing of this proposal Computer Science department is considering migrating from Visual C++ version 6.0 to Visual C++ for .NET; however its result does not affect this study.

needed to complete each take-home programming project. Mr. McQuain will be consulted with regard to the composition of each subset.

Following the completion of roughly the first 60% of the course, those students using the *CSI Sandbox* programming environment will be migrated back to Microsoft Visual C++ version 6.0 (or Visual C++ for .NET) for the remainder of the course. Peter DePasquale will provide an optional out-of-class training seminar on using the environment for those students migrating to Visual C++ to help ease the transition.

Collection and use of programming data: Students using the *CSI Sandbox* programming environment to complete their course work (both take-home and in-lab programming activities) will have all of their programming data automatically collected while using the environment³. The data collection includes the following:

- date and time
- Internet Protocol (IP) address of the computer
- machine name of the computer
- name and version number of the operating system
- vendor and version number of the Java run time environment in use
- version of the *CSI Sandbox* software in use
- complete source code listing
- complete error list
- complete output generated
- PID of the user
- Subset configuration details

Data collection from the programming environment occurs in real-time over the Internet (if applicable) and is stored in a database on a machine located in the Department of Computer Science at Virginia Tech. Programming data collected from this portion of the investigation will be stored until the completion of the dissertation writing process, expected to be during the summer of 2003.

Programming data is not collected automatically for students using Microsoft Visual C++. However, each programming and lab-based project completed by the students is required to be submitted to an automated grading service housed in the department. Student programming submissions to this service are stored according to date and time, as well as the submitter's PID. If necessary for the data analysis of this investigation, the investigators may examine the various versions of the subject's submissions.

Collection and use of self-assessment:

Initial demographic assessment: During the first week of the semester, all incoming Computer Science freshman will be surveyed to obtain various demographic data, programming experience, and general computer skills/proficiency information. These surveys will be performed by distributing OPSCAN forms to all Computer Science students participating in *CS 1104: Introduction to Computer Science* sections (2). In this fashion we will (in the optimal number of course sections) be able to gather data on all students participating in the CS1044 course, as well as the additional freshmen not enrolled in CS1044 (those who tested out, or were placed in *CS1114: Introduction to Problem Solving and Computing*), thereby capturing data from all freshmen Computer Science students.

Post-programming project meta-cognitive assessments: Following each take-home programming project in CS1044, we will ask the students to complete a short survey on their most recent experience with the programming environment they have utilized to complete their work. This series of surveys will seek to elucidate the students' perceptions of their environment, new functionality

³ Visual C++ does not support such data collection capabilities.

that they have learned to utilize, their interaction with the built-in help systems, where they obtained the most help, and how much time and effort went into completing the programming project.

Data collected by this portion of the project will be kept confidential by the investigators and will only be shared with the course instructor in an anonymous fashion, protecting the student's anonymity. The assessments (both demographic and post-programming project) will be correlated with other student-specific data collected in this investigation, but the resulting data will be anonymous. The assessment data will be destroyed at the completion of the dissertation writing process, expected to be during the summer of 2003.

Collection and use of academic work/grades: All academic work completed for the CS1044 course will be collected and evaluated by the investigators during this study. This includes (but is not limited to):

- Graded results from homeworks and quizzes
- Graded exams (mid-term and final)
- Auto-graded results from take-home programming projects
- Hand-graded results from take-home programming projects
- Auto-graded results from closed-lab programming projects
- Overall course grades

All coursework collected by this portion of the investigation will be stored securely (either electronically or physically) and viewable only by the course instructor, additional teaching assistants (as needed to perform their traditional activities), the project investigators, or the student who completed the work. Upon completion of the dissertation writing process, all data will be returned to the course instructor for disposal or further storage, as per department/university policy.

Collection and use of anecdotal data: Anecdotal data may also be collected during the period of this project. This will come in the form of e-mails (both solicited and unsolicited) submitted to the investigators by the students, verbal anecdotes related to the investigators in and out of the classroom (transcribed by the investigators), and transcriptions of group electronic communication mediums employed normally in the administration of the course (LISTSERVs and web-based message boards).

All anecdotal data collected during this project will be destroyed at the completion of the dissertation writing process, expected to be during the summer of 2003.

Payment for participation: Since this is an unfunded effort, there will be no payment for participation in the project. No extra course credit or points will be given to the participants of this study.

Risks and Benefits

See enclosed Informed Consent form.

Confidentiality/Anonymity

See enclosed Informed Consent form.

Informed Consent

Enclosed.

Biographical Sketches

Peter J. DePasquale
(540) 961-2133
Biographical sketch for the IRB

Peter J. DePasquale is currently a Ph.D. Candidate in the Computer Science Department at Virginia Tech. He holds Bachelor's and Master's degrees in Computer Science from Villanova University as well as a Master's degree (in Computer Science) from Virginia Tech. He has worked as a software engineer and systems administrator for Sonalysts, Inc.

Peter's research interest area revolves around computer science education and the use of interactive tools in support of the computer science curriculum.

Dr. John A. N. Lee
(540) 231-5780
Biographical sketch for the IRB

John A.N. Lee is Professor of Computer Science at Virginia Tech, Blacksburg, VA. Dr. Lee moved to the USA in 1964 at the University of Massachusetts, he initiated the Computer Science program, was the first department head, and in the meantime developed compiler software for first time-sharing system for CDC machines (BASIC, FORTRAN, APL). He wrote the first US textbook on compiler development - The Anatomy of a Compiler (1967) and first textbook on formal languages Computer Semantics (1971). He moved to Virginia Tech in 1974 to assist in the development of graduate program, and extend the undergraduate program. He has served as the chairman of the Undergraduate Program for many years, and has developed many new courses including "Professionalism in Computing", the WWW site for which is shared by many institutions.

Throughout these academic appointments he represented the Association for Computing Machinery (ACM) on American National Standards Committee X3 for 20 years, leading the development of Numeric Representation and BASIC standards; he participated in the standardization of PL/I, and oversaw standardization of Ada for the Department of Defense. He received a Certificate of Appreciation from the Computer and Business Equipment Manufacturers Association in October 1971, and a Certificate of Distinguished Service, from the US Department of Defense in October 1983 for his work on the standardization of the programming language Ada. He was designated a "Pioneer of Information Processing Standards", at the 25th Anniversary Meeting of the Accredited Standards Committee X3, in 1986.

Within ACM he served as the Chair or member of the standards committee for over 20 years, was elected as a ACM Council member, and Vice President. He received the ACM Outstanding Contribution Award in 1981, and Certificates of Recognition of Service in 1980, 1985, 1986, and 1989. In 1993 he was named as the recipient of the Distinguished Service Award by the Association for Computing Machinery for his service to ACM and the computer community through his innovative contributions to computing standards, the history of computing, and the development of professionalism. He was named as a Fellow of the Association for Computing Machinery in 1993.

His book of biographies of Computer Pioneers was published by the IEEE Computer Society Press (Los Alamitos, CA). He is currently working on the second edition.

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY**Informed Consent for Participants of Investigative Projects**

Title of Project: The Effect of a Simplified Programming Environment and Programming Language Subsets on Novice Computer Programmers

Investigator(s): Peter J. DePasquale, Ph.D. Candidate; Dr. John A.N. Lee, Professor

I. The Purpose of this Research/Project

This study is an examination of the effect of the application of both a simplified programming environment and the introduction of programming language subsets on novice students in a first year programming course. The project detailed in this proposal seeks to collect data to determine the following:

- Does a simplified programming environment (not designed for professional developers, less graphical user interface components, less complexity) yield quantifiable changes in student performance in a novice programming course?
- Does the application of programming language subsets yield quantifiable changes in student performance in a novice programming course?

Data collected by the proposed project will be used by Peter DePasquale in support of his Ph.D. dissertation in Computer Science.

II. Procedures

Selection of students: The students participating in this investigation will be drawn from the CS1044 course offerings for Computer Science majors this Fall (2002). Currently there are two lecture sections offered, and it is anticipated that roughly 240 students will be enrolled in the course.

Composition of project groups: Each student will be placed in one of three categories for the purposes of this investigation. The categories serve to dictate which programming environment the student will use through the first 60% of the course. The three environments being used are Microsoft Visual C++ (version 6.0 or .NET), *CS1 Sandbox* (without language subsets), and *CS1 Sandbox* (with language subsets).

Collection and use of self-assessment:

- **Initial demographic assessment:** During the first week of the semester, all incoming Computer Science freshman will be surveyed to obtain various demographic data, programming experience, and general computer skills/proficiency information.
- **Post-programming project meta-cognitive assessments:** Following each take-home programming project in CS1044, we will ask the students to complete a short survey on their most recent experience with the programming environment they have utilized to complete their work.

Collection and use of academic work/grades: All academic work completed for the CS1044 course will be collected and evaluated by the investigators during this study. This includes (but is not limited to):

- Graded results from homeworks and quizzes
- Graded exams (mid-term and final)
- Auto-graded results from take-home programming projects
- Hand-graded results from take-home programming projects
- Auto-graded results from closed-lab programming projects

- Overall course grades

Collection and use of anecdotal data: Anecdotal data may also be collected during the period of this project. This will come in the form of e-mails (both solicited and unsolicited) submitted to the investigators by the students, verbal anecdotes related to the investigators in and out of the classroom (transcribed by the investigators), and transcriptions of group electronic communication mediums employed normally in the administration of the course (LISTSERVs and web-based message boards).

Payment for participation: Since this is an unfunded effort, there will be no payment for participation in the project. No extra course credit or points will be given to the participants of this study.

III. Risks

There are no known risks to the physical well-being of the subjects during this study.

IV. Benefits of this Project

Your participation in this project will provide information that may be used to improve the usability of future software development environments, which would be used in courses such as CS1044. No guarantee of benefits has been made to encourage you to participate, and you are not being asked to do any additional work that everyone else will be doing in this course by participating in this study. You may receive a synopsis summarizing this research when completed. Please provide a self-addressed envelope with the experimenter and a copy of the results will be sent to you.

V. Extent of Anonymity and Confidentiality

The results of this study will be kept strictly confidential. Your written consent is required for the researchers to release any data identified with you as an individual to anyone other than personnel working on the project. The information you provide will have your name removed and only a subject number will identify you during analyses and any written reports of the research.

VI. Compensation

Your participation is voluntary and unpaid.

VII. Freedom to Withdraw

You are free to withdraw from this study at any time for any reason. If you choose to withdraw, your data will not be utilized as part of the results of this study, and the choice of programming environments you utilize will be left to the course instructor. Withdrawal from the study will be implied if you officially withdraw from the CS1044 course. Withdrawal from the study does not imply official withdrawal from the CS1044 course.

VIII. Approval of Research

This research project has been approved, as required, by the Institutional Review Board for Research Involving Human Subjects at Virginia Polytechnic Institute and State University, and by the Department of Computer Science.

IX. Subject's Responsibilities

I voluntarily agree to participate in this study. I have the following responsibilities:

- Participate in the CS1044 course to the best of my abilities.
- Utilize only the programming environment I am instructed to use for both take-home programming projects and the in-lab programming activities.
- Do not collaborate with any other student currently or previously enrolled in CS1044 for completing my programming projects.

X. Subject's Permission

I have read and understand the Informed Consent and conditions of this project. I have had all my questions answered. I hereby acknowledge the above and give my voluntary consent for participation in this project.

If I participate, I may withdraw at any time without penalty. I agree to abide by the rules of this project.

Signature

Date

Should I have any questions about this research or its conduct, I may contact:

Investigator: Peter J. DePasquale
Ph.D. Candidate, Department of Computer Science

Phone: (540) 231-5914
Email: pjdepasq@vt.edu

Investigator: Dr. John A.N. Lee
Professor, Department of Computer Science

Phone: (540) 231-5780
Email: janlee@cs.vt.edu

Review Board: David M. Moore
Research Compliance Office, CVM Phase II (0442)

Phone: (540) 231-4991

Subjects must be given a complete copy (or duplicate original) of the signed Informed Consent.

Appendix B

Demographic Survey

CS1 Sandbox Project - Demographic Survey

Your responses on this survey will be kept completely anonymous. Please complete ALL questions, as well as the **Name and ID Number** portion on the upper right section (needed for data correlation with those in CS1044). No professor or GTA (except for Pete DePasquale) will see your results, and they will be destroyed at the completion of his Ph.D. dissertation. Results will be made public, but anonymity will be preserved. This survey will have no impact on any grade in this course or Honor Court case, past, present, or future.

Participation is voluntary, and appreciated. Please be as honest and accurate as possible. Only fully completed surveys will be eligible for the prize drawing.

Peter J. DePasquale is conducting this survey as part of his Ph.D. research.

Question:	Response(s)
1) What is your academic level?	1. freshman 2. sophomore 3. junior 4. senior 5. transfer Other: _____ (please describe)
2) What is your gender?	1. male 2. female
3) What is your age (range)?	1. 17 2. 18 3. 19 4. 20 5. 21 6. 22-30 7. 30+
4) Are you entering college having just completing high school?	1. yes 2. no

For questions 5-16, please categorize your programming experience for each of the languages as follows. Please answer for EACH language. (HTML is not a programming language).

1. None – never used language
2. Slight – can read code and know a little about this language
3. Some – attempted to write my own program(s) in this language
4. Good – had programming classes in high school/trade school/other formal training
5. AP – took (and passed) AP-level programming courses in high school/trade school
6. Better – worked summer jobs as a programmer using this language
7. Pro – worked full time (1 year or more, 40 hours per week) as a programmer or software developer (web development does not count!) using this language

Application Languages:	Web Development:	Misc. / Other
5) Basic	12) Perl	15) Macro programming
6) C	13) JavaScript / ASP	16) Other _____
7) C++	14) PHP	(please name)
8) Java		
9) Fortran		
10) Pascal		
11) Visual Basic		

17) Please list your combined SAT score, if you took the SAT: _____

18) Please list your combined ACT score, if you took the ACT: _____

Appendix C

CS1044 Course Syllabus – Fall 2002

CS 1044 Intro. to Programming in C++

Course Policy Statement for CS Majors

Instructor: William D McQuain E-mail: mcquain@cs.vt.edu
631 McBryde Hall Office hours: 2:00—4:00 MW

Purpose: To provide an introduction to structured procedural programming using elementary features of the C++ language.

Prerequisites: No formal prerequisites. However, competence in the Windows environment (editing and copying files, navigating directories with Explorer, etc.) and previous problem solving experience are highly recommended.

It is required that you have a valid University PID and know your password in this course.

Text: For general reference on the C++ language, programming examples, etc: *Programming and Problem Solving in C++*, 3rd Ed., by Dale, Weems and Headington. An inexpensive copy of the course notes is available through A-1 Copy Center (University Mall) — these are considerably cheaper than printing the notes yourself.

TAs: The Teaching Assistants for this course are listed on the course website. Their office hours will be announced and posted on the course website as soon as possible.

Course Website: (<http://courses.cs.vt.edu/~cs1044/fall02/mcquain>) The course website will include copies of the course contract (this document), pertinent department policy statements, office hours, test dates, homework assignments and programming project specifications as available, and timely announcements. You are advised to consult the website on a regular basis, especially if you are foolish enough to skip class regularly.

The course website also has links to other useful information, including brief tutorial introductions to the Visual C++ editor and debugger, example programs, koofers, updates to the course notes, and a link to the homepage for the Curator System (<http://www.cs.vt.edu/curator/>).

Assignments: Your grade will be based on lab and homework assignments, two tests, a final exam, and a number of programming projects, weighted as follows:

Item	Weight	Tentative Dates
In-lab Assignments and Homework	20%	Varied
Out-of-lab Projects and Software Engineering	40%	Varied
Tests (2)	8%, 12%	TBA
Final Exam	20%	TBA

Homework Assignments: Some homework assignments will consist of multiple-choice questions that relate to the lectures, course notes, projects and reading. Since I will use the same format for the questions on the tests and final exam, doing and understanding these homework assignments will help you prepare for those tests.

Programming Projects: The out-of-lab programming projects must be implemented in ANSI C/C++, as described in the course notes. You may use any ANSI conformant compiler you like, however your programs will be compiled and tested using Visual C++ .NET, running on Windows NT/2000.

There will be two kinds of programming assignments in the majors-only sections of CS 1044: in-lab and out-of-lab. Unless specified otherwise by the lab instructor, both kinds will be subject to the same Honor Code requirements given later in this document.

The in-lab assignments will vary considerably in complexity. Some will require completing given programs, some designing and implementing small to medium programs, some carrying out experiments involving testing or debugging code, and some will not involve actual programming at all.

Each of you will be attending a specific lab session. Each session will be required to use a specific development environment for the in-lab assignments. These include Visual C++ .NET, and two variants of an instructional "sandbox".

The Visual C++ .NET compiler and the "sandbox" are the only ones supported for this course. That means that neither the TAs nor I will answer questions about the use of any other compiler, except Visual C++ version 6.0. The Visual C++ compiler and the "sandbox" are installed on a number of Windows PCs in various computer labs around campus. If you are using another compiler for the out-of-lab projects, you should test each of your programming projects in the lab prior to submission. There are known (and unknown) compatibility issues with other C++ compilers, including Visual C++ 6.0 and especially g++ on Linux. If you choose to develop with any compiler than Visual C++ .NET, it is your responsibility to discover and deal with those issues. We will make no allowances in grading because you have chosen to use a different compiler.

All the out-of-lab programming projects will be subjected to runtime testing using the Curator System. See the Curator homepage for details, including the instructions you will need in order to submit assignments to the Curator. Be sure to read the *Student Guide to Submitting* in the course note pack—it contains the answers to most of the questions students have about the automated grading system. The *Student Guide* also contains information about how the Honor Code applies when using the Curator; be sure to read and follow the guidelines given there.

A number of the out-of-lab programming projects will also be graded for adherence to good software engineering principles, including documentation, design, conformance to the stated specification, and programming style. Each project specification will include explicit guidelines that you will be expected to follow. The TAs will grade your (first) submission to the Curator that received the highest score, and e-mail you the results. Note that if you make an incomplete submission (e.g., omitting required documentation) and that receives a perfect score, then the TAs will evaluate that incomplete submission. There will be no exceptions to this policy. If you do not make a submission for a project, then you will receive a zero for software engineering for that project.

Tests: Your score on the final exam will automatically replace the lower of your two test scores, if it is an improvement. You must bring your Va Tech ID card to the tests and final exam! Because the tests and final exam are multiple choice and are scored via machine, also bring a number 2 pencil and a good eraser.

Grading Policies: This course is largely devoted to the development of skills in structured programming, as reflected in the relatively heavy weight given to the programming assignments. You will be expected to produce programs which are not only functionally correct, but also well-structured, well-documented and readable. The Computer Science Department Documentation Standards, described in *Elements of Programming Style*, will be enforced on any programming assignments that are human-graded (a copy is included with the course notes).

Backups: It is your responsibility to maintain an up-to-date backup copy of each programming project (that is in addition to the copy you submit). The hard drives of the lab machines are recloned periodically, so don't try to leave a backup there! Keep a spare copy of all the relevant files for each project on a Zip disk or a CD-R in case your assignment is mislaid. (Floppy disks are notoriously unreliable.)

Late Work: Each programming project and homework assignment will have a due date and time and will include instructions for submission. Except in the very rare case that an extension is granted, late submissions will incur a penalty of 20% per day, and will not be given any credit if submitted after graded assignments or solutions have been released. Any request for an extension must be made at least 24 hours prior to the due date.

Plan your time carefully for the programming projects, especially if you will be using computers in the campus labs — you may be competing with other students for scarce resources, so don't put things off until the last minute.

Note well: delays resulting from machine availability, lab schedules, hardware failures or your failure to maintain a backup of your work do not merit an extension.

Statute of Limitations: Any questions or complaints regarding the grading of an assignment or test must be raised within two weeks after the score or the graded assignment is made available (not when you pick it up).

Absences: If a serious illness prevents you from taking any of the tests, send a friend with a note describing your condition or notify me before the day of the test. Also, to establish a valid excuse for an illness you must get a note from a physician or the University infirmary. Before missing a test for any reason, you must make every effort to discuss the problem with me before the day of the test. Excuses other than an illness must be reported to your Dean's office so that they can send me a written explanation of the absence. If you need to be away for an official University event, this must be cleared with me in advance. Without a valid excuse, no makeup tests or exam will be given!

Grade Scale: Final grades will be set according to the usual 10-point scale; i.e., 90% guarantees at least an A-, 80% at least a B-, etc.

Honor Code: An exhaustive list of Honor Code violations would be impossible to present here, but among other things, each of the following is a flagrant violation of the Virginia Tech Honor Code, and violations will be dealt with severely (Honor Court):

- Working with another student to derive a common program or solution to a problem. Unless explicitly stated in a specification, there are no group projects in this course.
- Discussing the details required to solve a programming assignment. You may not share solutions.
- Copying source code (programs) in whole or in part from someone else.
- Copying files from another student's disk even though they might be unprotected.
- Editing (computer generated) output to achieve apparently correct results.
- Taking another person's printout from a lab printer, remote rprint printer, trash can, etc.

It is acceptable to discuss with classmates a programming assignment in a general way, i.e., to discuss the nature of the assignment. In other words, you may discuss with your classmates what your program is required to accomplish but not how to achieve that goal using C++. In no way should the individual statements of a program or the steps leading to the solution of the problem be discussed with or shown to anyone except those people cited in the following statement.

Feel free to discuss the homework assignments and your program source code with the teaching assistants assigned to CS 1044, the instructor, or the free tutors provided by UPE. The discussion of your program source code must be limited to these people. Note that this specifically excludes discussions of your program source code with other students (even if they are not enrolled in CS 1044), or with tutors except for those named above. Privately hired tutors are not an exception to this requirement, nor are athletic or other tutors provided by the University.

Copies of all submitted work are retained indefinitely by the Department. Submitted programs are subjected to automated analysis for detection of cheating.

If you have any question as to how the Honor Code applies to this class, remember that:

- Any work done in this class must be done on an individual basis.
- Credit will be given only for work done entirely on an individual basis.
- Do not make any assumptions as to who can provide help on a programming assignment.
- All submitted work is archived. All submitted programs will be subjected to automated cheat analysis.
- Evidence indicating the violation of the policy stated above will be turned in directly to the Honor Court.
- It is much easier to explain a poor grade to parents or a potential employer than to explain an Honor Court conviction.

In addition, the Honor Code statement included in the *Student Guide to the Curator* is in force for this class.

The Honor Code will be strictly enforced in this course. All assignments submitted shall be considered pledged graded work, unless otherwise noted. All aspects of your work will be covered by the Honor System. Honesty in your academic work will develop into professional integrity. The faculty and students of Virginia Tech will not tolerate any form of academic dishonesty.

Appendix D

Closed-Laboratory Specifications

D.1 Laboratory #1

CS1044 Fall 2002**Lab 1****Name:** _____**Grade:** _____**Part 1: Short Answer****(30 points via hand grading)**

Identify the following two properties of algorithms

1. _____ Algorithm must complete after a finite number of instructions have been executed.
2. _____ All instructions must be able to be performed. Illegal operations (division by 0) are not allowed.
3. Polya's Four-Step Process: Name the steps
 - a. _____
 - b. _____
 - c. _____
 - d. _____
4. Name the three language levels discussed in class:
 - a.
 - b.
 - c.
5. _____ is a documentation tool that enables the designer to keep track of the relationships among subproblems throughout the software refinement process.
6. Name the categories of programming errors:
 - a. _____ (compilation) errors
 - b. _____ errors
 - c. _____ (runtime) errors
 - d. _____ errors
7. _____ are rules that specify how valid instructions are written.

Part 2: Short coding**(10 points via hand grading)**

Use your designated programming environment (.NET, Sandbox with subsets, or Sandbox without subsets) to enter the following program. Substitute the appropriate information about yourself where designated (removing the dollar signs). Enter the code carefully and be sure to check for errors. Enter the comments as well!

```
//
// Program Author: $YOUR NAME$
// Date of submission: $TODAY'S DATE$
// Lab Section: $.e.g. Wednesday 1:25pm$
//
#include <fstream>
using namespace std;

int main () {
    // Declare the output stream and open the file.
    ofstream outFile;
    outFile.open("lab1Out.dat");

    // Initial preamble for output
    outFile << "Welcome to CS1044 - Fall 2002" << endl;
    outFile << endl << endl;

    // Student specific data output
    outFile << "My name is: " << "$add your name here$" << endl;
    outFile << "I am in the " << endl;
    outFile << "    $8:00AM or 10:10AM$ section of 1044." << endl;
    outFile << "I am using the following programming environment: " << endl;
    outFile << "    $Visual C++ .NET or Sandbox$" << endl;

    // Close the output file
    outFile.close();

    // Return from function
    return 0;
}
```

This is end-ell
not end-one!
Be sure type
this carefully!

What is the output produced by your program? Print it **legibly** in the section below:

Part 3: Submit to curator**(60 points via Curator grading)**

When you have completed and tested the program you created in Part 2, log into the curator system (refer to class website for link) and submit it for grading. Remember that you should select the curator section that corresponds to your lab (CS104401, CS104402, CS104403). Be sure to select **LAB1** as the name of the project.

You may resubmit this programming task up to 5 times for the largest possible grade (60 points).

D.2 Laboratory #2

CS1044 Fall 2002

Lab 2

Name:

Grade:

Part 1: Fill in the blanks

(10 points from hand grading)

Complete the following program by filling in the missing C/C++ code.

```
// Converts a time interval measured in seconds to its
// hours:minutes:seconds form.

#include <iostream>
using namespace std;

int main ()
{
    int TimeInSeconds,
        RemainingSeconds,
        Minutes,
        Hours;

    // Prompt for time interval and read it in.
    cout << endl << "Enter the time interval (in secs): ";
    cin >> TimeInSeconds;

    // Convert to hours:minutes:seconds form.
    Hours = TimeInSeconds _____ 3600;           // Determine the number of hours

    RemainingSeconds = TimeInSeconds % _____; // Determine the remaining seconds

    Minutes = _____ / 60;                       // Determine the number of minutes

    RemainingSeconds = RemainingSeconds _____ 60; // Determine the remaining seconds

    // Display in hours:minutes:seconds form.
    cout << _____ << ":" << Minutes << ":" << RemainingSeconds << endl;

    return 0;
}
```

Part2: Five and Dime

(10 points from hand grading, 80 points from the Curator)

You can use integer arithmetic to compute the mix of coins returned by a vending machine. The key is to represent all monetary amounts in cents as `int` values, rather than as fractions of a dollar.

Step 1: Enter and complete the program provided below so that the program calculates the number coins returned from a vending machine as change from a purchase. Assume that the cost of the purchase and the amount of money put in machine are measured as cents and are stored in variables of type `int`. Furthermore, assume that these values are multiples of 5 cents (no pennies). The program simulates a user providing one dollar to the cashier for the purchase and that your program will calculate how much change needs to be returned to the user. Be sure to save your program in a file named: *change.cpp*.

When calculating the change amount, the program will return as many larger coins as possible. That is, if the amount to return is 80, don't return 8 dimes, return 3 quarters, and one nickel. To do this, we begin our calculations from the large coins and work down to the smaller ones.

```

#include <fstream>
using namespace std;

int main () {
    // Declare our file streams (input and output files)
    ifstream inFile;
    ofstream outFile;

    // Declare the variables we need (you add needed variables!)
    int totalCost,
        changeInCents,

        ;

    // Open the input and output files for reading/writing
    inFile.open("changeInput.data");
    outFile.open("changeOutput.data");

    // Read cost from input file
    inFile >> totalCost;

    // Perform calculations (you provide entire code here!)

    // Output the results (you complete the parts that are missing!)
    outFile << _____ << " cents change" << endl;
    outFile << _____ << " quarter(s)" << endl;
    outFile <<
    outFile <<

    // Close the file streams
    outFile.close();
    inFile.close();

    // Return a zero and finish program
    return 0;
}

```

**Remember
from last week,
this is end-ell,
not end-one!**

Program Input: The actual cost of the purchase (in cents), located in a file named *changeInput.data*. You will create this file and provide an integer value not great than 95, and not less than 0, and whose value is a multiple of 5 (thus avoiding providing change with pennies!). The file should be created with a simple text editor program (notepad works well, Word is overkill and does not result in a simple text file!).

The file should contain only one line with one integer value on the line. This value is the actual cost of the purchase (not the amount of change returned). Consider testing your program with several (5 or more) values producing different sets of output to ensure that all of the possible cases and conditions are examined!

Program Output: Total change returned, located in a file named *changeOutput.data*. You will produce the file programmatically (via your program). Be sure you have the output file name correct, as the curator looks only for this file when grading your submission.

The output to produce will be of the following form:

- A single line indicating the total amount of change to return, followed by a newline character.
- A single line indicating the number of quarters returned, followed by a newline character.
- A single line indicating the number of dimes returned, followed by a newline character.
- A single line indicating the number of nickels returned, followed by a newline.

For example, if the input file contained a value of 70, the output produced would be:

```
30 cents change
1 quarter(s)
0 dime(s)
1 nickel(s)
```

Remember that a newline is produced by outputting a '\n' character or the value endl!

Step 2: Complete the following test plan for your program. Be sure you validate each of your results with the test plan and **put a check mark in the box** if your results match. If your results do not match, correct your program and then recheck your results.

Test plan for *change.cpp*

Test Case	Sample Data	Expected Result	Checked
No change		0 cents change 0 quarter(s) 0 dime(s) 0 nickel(s)	
Only quarters returned	25	75 cents change 3 quarter(s) 0 dime(s) 0 nickel(s)	
At least one of each coin returned		40 cents change 1 quarter(s) ___ dime(s) ___ nickel(s)	
Only a dime returned		___ cents change 0 quarter(s) 1 dime(s) 0 nickel(s)	

Once your program has produced the output file, view its contents with a simple editor like the Windows notepad to ensure it is correct. The Curator system grades your output on correctness, so it's important to match the output exactly. Note the presence of the '(s)' following the coin denomination, the spacing, and the spelling!

Step 3: Submit your program to the Automatic Grader. You should open a web browser on your system and point it to the course web site (<http://courses.cs.vt.edu/~cs1044>) to locate the curator interface.

Be sure to select the proper section for the lab you are in (or the Friday section if you are making up a lab). Do not submit to the "Out-of-Lab" section. Once logged in, click the *Submit link*, and choose the Project **LAB2**. Then upload your file via the *Browse... button* and the *Upload button*, and then the *Home link* to check your grades. It may take up to 5 minutes for your project to be graded, so be patient and keep checking.

You have a maximum of 5 submissions for this program, thus you may submit more than once. The highest graded submission will contribute to this lab's overall grade.

D.3 Laboratory #3

CS1044 Fall 2002

Lab 3

Name:

Grade:

Part 1: Show me the money...

(90 points from curator grading)

In this lab, you will complete your own program (no source code fragments this week!). The program will open an input file (*moneyInput.data*), read both string and monetary data, outputting (to a file named *moneyOutput.data*) some of the strings (ignoring others) and summing the monetary amounts read in from the input file.

Input file: (moneyInput.data)

The input file will take the following form:

1. The first line will contain a person's name (first name, middle initial, and last name).
2. The second line will contain one of three descriptions of the person ("A good friend", "A business associate", or "A close relative").
3. The third line will be blank.
4. The fourth line will contain two monetary values, separated by the sequence: a tab, an asterisk, and a tab (i.e.: \$55.55 * \$32.00).
5. The fifth line will contain a single monetary value.
6. The sixth line will contain two monetary values in the same format described in #4.
7. The seventh and eighth lines will each contain single monetary values.
8. The eighth line of the input file is the last line.

An example of the input file would be:

```
Peter J. DePasquale
A business associate

$66.77 * $33.00
$12.00
$10.00 * $0.33
$25.50
$50.25
```

Each monetary value read from the file will be summed together to produce a final, complete total of the values. No monetary input value will be invalid, and each will be in the form of *XXX.XX*. That is, each value will be preceded by a dollar sign, one or more dollar digits, a period, and two cents digits (no more, no less). Cent values less than 10 will be preceded by a zero (e.g. \$10.00, or \$10.05). Dollar values equal to zero will be represented by a single zero (e.g. \$0.55). The asterisk in the lines with two monetary values is to be ignored.

Output file: (moneyOutput.data)

The output of this program will be the following:

1. The first (and only) line will contain the string "*Y (type) owes you \$XX.XX*", where *Y* is the person's name read from the first line of the input file, *type* is the third word from the second line of the input file, and the monetary amount is the sum of all of the monetary values seen in the input file.

An example of the output file (given the input file above) would be:

```
Peter J. DePasquale (associate) owes you: $197.85
```

Note that the monetary value printed in the output file may be larger than two digits long. This is acceptable (and likely to happen). When printing this value, print the dollar sign, no spaces, and then the dollar value (without leading zeros), the period character, and then the cents (two digits).

When you have completed the program, submit it to the curator for grading. Be sure to log into the correct section and submit it as the project labeled "**Lab3**". You have a maximum of five submissions for this lab.

Other restrictions:

The use of user-defined functions, iteration (for, while, do loops), and selection statements (if, switch) are expressly forbidden in this lab. Prior to leaving the lab, you will show a teaching assistant your submitted curator code (via the curator interface) and have them initial in the space below that you have not used any of these items.

Part 2: Show me the code...**(10 points from hand grading prior to leaving)**

Prior to leaving the lab, but after submitting to the curator, you will call over a teaching assistant and show your submitted source code (via the curator interface). Assuming the teaching assistant does not see a violation of the restrictions (see above), they will initial the lab sheet below and take your sheet.

If you have violated the rules, you must correct the source code prior to leaving the lab. Additional submissions will not be granted for this purpose, and if necessary (you are out of submissions, but violated the restrictions in attempting to complete the project) we will deduct 30 points from your grade.

Teaching assistant initials:

Miscellaneous links:

Course web site: <http://courses.cs.vt.edu/~cs1044/fall02/mcquain>

CS1 Sandbox: <http://sandbox.cs.vt.edu/~sandbox>

Curator: <http://www.cs.vt.edu/curator>

For those using subsets in conjunction with the CS1 Sandbox program, please use the subset entitled "Project 1/Labs 3 and 4" during this lab!

D.4 Laboratory #4

Spring 2002 CS 1044 Lab

WEEK 4

Name: _____

Grade: _____

Part 1: Logic brush-up

(10 points from hand grading)

Consider the following code fragment:

```
bool C, D;
// code that assigns values to C and D
if ( C || !D ) {
    cout << "one" << endl;
}
else if ( D ) {
    cout << "two" << endl;
}
else {
    cout << "three" << endl;
}
```

For each of the output values below (1st column), specify the values of variables C and D that will produce the output. Specify each of the values of C and D as pairs in the form of: (C=true, D=false). Note that there are zero or more pairs for each output value (in other words, provide ALL possible pairs).

Value output	Values of C and D
"one"	
"two"	
"three"	

Part 2: Chinese Calendar**(10 points from hand grading)**

In the Chinese calendar, every year is associated with a particular animal. The 12-year animal cycle is a rat, ox, tiger, rabbit, dragon, snake, horse, ram, monkey, rooster, dog, and boar. The year 1900 was the Year of the Rat; thus 1901 was a Year of the Ox, and 1912 was another Year of the Rat. If you know in what year a person was born, you can compute the offset from 1900 and determine the animal associated with that person's year of birth.

Complete the following test plan for the calculation of the animal that corresponds to a year.

Test plan for Chinese Calendar program

Test Case	Sample Data	Expected Output
Base Year	1900	
Mid-cycle Year		1942 is the year of the horse.
Current Year		
Your birth Year		
Maximum Year		

Complete the expression below that determines which animal corresponds to a given year. You must assume that the rat is animal number 0 (and the ox is animal number 1) for this equation to work properly.

$$animalNumber = (year - \underline{\hspace{2cm}}) \% \underline{\hspace{2cm}}$$

Part 3: The program**(80 points from curator grading)**

Step 1: Create a program that determines the animal corresponding to an input year. Name your source code file *chineseNY.cpp*. Input for this program will come from an input file name *yearInput.data*, which contains a single line that contains two years separated by a single tab character.

For example, the following is sample input file for this program:

```
yearInput.data
1900 1968
```

For each year value read, determine if it is valid (it falls between 1960 and 2010, inclusively) or is invalid (it falls outside of the specified range).

Step 2: The output of your program must be to the file named *yearOutput.data*. This output file will contain the following output:

1. the first line will contain your name, and
2. the remaining lines will contain one of the two following phrases, depending on the validity of the data:
 - a. For valid year values (see criteria for validity above), the phrase "XXXX is the year of the YYYY." is printed. Here, XXXX is the full value of the year given by the input file, and YYYY is the animal name (in lower case).
 - b. For invalid year values, the phrase "XXXX is an invalid year value." is printed. Here too, XXXX is the full value of the year given by the input file.

Note carefully the format, spelling and capitalization of the output line. For example, for the input file shown above, the output file would be:

yearOutput.data

```
Peter DePasquale  
1900 is an invalid year value.  
1968 is the year of the monkey.
```

Step 3: Submit your completed program to the Automated Grader for credit. Be sure to select **Lab4** as the name of the project and submit only your source code file (*chineseNY.cpp* file) for grading. You have a maximum of 5 submissions for this project.

Miscellaneous links:

Course web site: <http://courses.cs.vt.edu/~cs1044/fall02/mcquain>

CS1 Sandbox: <http://sandbox.cs.vt.edu/~sandbox>

Curator: <http://www.cs.vt.edu/curator>

For those using subsets in conjunction with the CS1 Sandbox program, please use the subset entitled “Project 1/Labs 3 and 4” during this lab!

D.5 Laboratory #5

Spring 2002 CS 1044 Lab

WEEK 5

Name: _____

Grade: _____

Part 1: Counting the characters

(100 points from Curator grading)

Your program this week will open an input file named *text.in* and count the following from the input file:

- The number of each of the following lower-case vowels (count each SEPARATELY): 'a', 'e', 'i', 'o', and 'u',
- The number of white space characters which include only the space, tab and new line characters (count all as one group), and
- The number of other characters such as letters (excluding vowels listed above), numbers, punctuation, and special characters excluding the white space characters counted above (count all as one group).

The contents of the input file may (and likely will) contain multiple new line characters. The last line of the input file will end with a new line character.

The output produced by this program will go the file *text.out* and will repeat the input string from the file, followed by the count of each of the items listed above, in the following format (where # designates an integer value):

```
Original text listed here first in the same
order and format as given by the input file.

      a      e      i      o      u      ws      other
-----
      #      #      #      #      #      #      #
```

For example, suppose we have the following input file (the '\n' characters are used to indicate new line characters, and the end-of-file marker is not shown):

```
Now is the time\n
for all good men\n
to come to the aid\n
of the party.\n
```

Our program would produce the following output file:

```
Now is the time
for all good men
to come to the aid
of the party.

      a      e      i      o      u      ws      other
-----
      3      6      3      8      0      16      30
```

To get you started, we provide the following code framework:

```
// Added necessary include statements here
using namespace std;

int main()
{
    char character; // Contains character most recently read from input
    ifstream inFile; // Declare variables for input/output streams (files)
    ofstream outFile;
    int numberAs=0, // Contains the # of 'a's discovered
        numberEs=0, // Contains the # of 'e's discovered
        numberIs=0, // Contains the # of 'i's discovered
        numberOs=0, // Contains the # of 'o's discovered
        numberUs=0; // Contains the # of 'u's discovered

    // Add more variables as needed!
    .
    .
    .

    // Open the input and output files
    inFile.open("text.in");
    outFile.open("text.out");

    inFile.get(character); // Read a character from the input source

    while (inFile)
    {

        // Determine character's type and increment appropriate counter

        // Output character to first part of output file

        // Read the next character from the input file

    }

    // Produce output detailing final totals

    // Close the files and return a 0 to exit the program.
    inFile.close();
    outFile.close();
    return 0;
}
```

The program should continue to read characters from the input file until no more characters exist.

Submit your program to the Curator for grading. Be sure to choose **Lab5** as the name of the project. You have a maximum of 5 submissions for this assignment.

Miscellaneous links:

Course web site: <http://courses.cs.vt.edu/~cs1044/fall02/mcquain>

CS1 Sandbox: <http://sandbox.cs.vt.edu/~sandbox>

Curator: <http://www.cs.vt.edu/curator>

For those using subsets in conjunction with the *CSI Sandbox* program, please use the subset entitled "Labs 5-7 / Project 2" during this lab!

D.6 Laboratory #6

Spring 2002 CS 1044 Lab

WEEK 6

Name: _____

Grade: _____

Part 1: Word Search

(100 points from Curator grading)

Given an input file named *Words.text*, your program will open the file, process the input searching for specific words, and printing the desired word to the output file named *Summary.text*.

The input file is comprised of an unknown number of lines of text. Each line of text will be of the following forms:

- **text line:** the line starts with an integer value (possibly negative, zero, or positive), and a series of words from the English language. Words are separated by **one or more** white space characters (defined this week as spaces and tabs),
- **sentinel line:** the integer value `-9999` on a line by itself signifying the end of the input to be processed. This line may be followed by invalid lines of text, described below,

For example, the following file is a valid input file: (the ‘`\n`’ characters are used to indicate new line characters, and the end-of-file marker is not shown):

```
5 There is a mouse in my house.\n
2 Press any key to continue.\n
-4 We hold these truths to be self-evident.\n
-9999\n
5 Once upon a time, there was a girl named Ines.\n
-6 She was a teaching assistant at Virginia Tech.\n
```

For each text line, your program will output the length of the N^{th} word on the line, and the N^{th} word itself, where N is defined to be the integer value found at the start of the line. If N is positive, the format of the line produced will be:

```
length: word
```

If N is negative or zero, then the message “Error: N ” should be printed to the output file, where N is the integer value read at the start of the line. The format of the line produced will be:

```
Error: value
```

Your program should stop processing lines from the input file when the end-of-file marker is seen, or the sentinel value has been read (whichever comes first). Input lines that follow the sentinel value should not be processed.

The first three lines of the output file should contain the following:

```
Programmer: <your name>\n
Word Hunt
```

Given the input file above, our program would produce the following output file:

```
Programmer: Peter DePasquale\n
Word Hunt\n
2: in\n
3: any\n
Error: -4
```

To get you started, we provide the following code framework:

```
int main() {  
    ifstream inFile;  
    ofstream Log;  
  
    // Open the files and write the header to the output  
  
    // Priming read  
    inFile >> wordPosition;  
  
    while ( inFile && _____ ) {  
  
        if ( _____ ) {  
  
        }  
        else {  
  
        }  
  
        // Read the next value  
    }  
  
    // Close files and complete processing  
}
```

Submit your program to the Curator for grading. Be sure to choose **Lab6** as the name of the project. You have a maximum of 5 submissions for this assignment.

Miscellaneous links:

Course web site: <http://courses.cs.vt.edu/~cs1044/fall02/mcquain>

CS1 Sandbox: <http://sandbox.cs.vt.edu/~sandbox>

Curator: <http://www.cs.vt.edu/curator>

For those using subsets in conjunction with the *CSI Sandbox* program, please use the subset entitled “Labs 5-7 / Project 2” during this lab!

D.7 Laboratory #7

Fall 2002 CS 1044 Lab

WEEK 7

Name: _____

Grade: _____

Part 1: Applying functions to the Character Counting Lab

(100 points from Curator grading)

Recall the lab program from two weeks ago. In it, we read in a file of text, a character at a time, and output the number of lowercase vowels, whitespace characters and “other” characters.

In this lab, we will apply a function to the program to abstract the identification of a character as a lowercase vowel ('a', 'e', 'i', 'o', and 'u'). The function you will author will contain a strict interface that we will define below. Once you have completed the project, the teaching assistant will examine the code you submitted to the Curator to ensure that you have adhered to this interface.

The function:

- shall return a Boolean (bool) value which is true if the character parameter is a space, tab, or newline. In all other cases, the function will return a false value,
- the function shall be named **isWhiteSpace**,
- the function shall accept a single pass-by-value parameter which is a character type.

Thus, the function should take the form of:

```
bool isWhiteSpace(char someCharIdentifier)
```

Recall that the input file (again named *text.in*) is in the form of one or more lines (each terminated by a new line character), which contains numbers, letters, white space (' ', '\t', '\n') and special characters (such as '\$', '#', etc.). An example of the input file follows (the '\n' characters are used to indicate new line characters, and the end-of-file marker is not shown).

```
Now is the time\n
for all good men\n
to come to the aid\n
of the party.\n
```

The output (a file named *text.out*) for this program will be the same as the previous lab. You will reproduce the input file exactly followed by a blank line. Next, a line of column headers will be presented, followed by a line of minus-signs acting as a separator line, followed by the results of the counted number of lowercase vowels, white space characters and other characters. An example of the output file is:

```
Now is the time
for all good men
to come to the aid
of the party.

-----
      a       e       i       o       u       ws       other
-----
      3       6       3       8       0       16       30
```

Consider that you should have already written 90% of this solution and it's in your Curator “account”. The fact is, you only need to relocate a small amount of code into the function you are creating.

NOTE: 100 points from this lab comes from the Curator solution, though you may be docked **100 points** for failure to solve the problem using the required function. **20 points** will be deducted from your score if you do not adhere to the function's interface (return value, name, and parameter type(s)/list).

Teaching assistant initials:

(more on back)

Miscellaneous links:

Course web site: <http://courses.cs.vt.edu/~cs1044/fall02/mcquain>

CS1 Sandbox: <http://sandbox.cs.vt.edu/~sandbox>

Curator: <http://www.cs.vt.edu/curator>

For those using subsets in conjunction with the *CSI Sandbox* program, please use the subset entitled “Labs 5-7 / Project 2” during this lab!

D.8 Laboratory #8

Fall 2002 CS 1044 Lab

WEEK 8

Name: _____

Grade: _____

Part 1: More fun with functions

(100 points from Curator grading)

The program:

In last week's lab, we introduced a function into our program. That function returned a Boolean value, and was passed a single character parameter (passed by value). In this lab, we'll get a tad more complex...

Here is a sample input file, named *rectangle.data*, for the program. The file specifies the coordinates of two opposite corners of a rectangle that is to be drawn on a video screen. Note that in this coordinate system, the x-axis runs along the top of the screen and the y-axis runs down the left side of the screen, with the origin at the upper-left (or northwest) corner of the screen. So, y-values increase as you go down, not up.

The first line specifies the northwest corner and the second the southeast corner of the rectangle. Each corner is expressed using the customary mathematical notation for a point, and is preceded by a descriptive label that ends with a colon character ':'. (The labels will always be the ones shown below, but that shouldn't matter.):

```
NW corner: ( 39, 155)␣
SE corner: ( 80, 248)␣
```

The program will read the values, and calculate the area of the rectangle that is created by the points. There will be five lines of output in the output file produced (named *rectangle.out*). They are:

```
Programmer: Peter DePasquale␣
CS 1044 Lab 8␣
␣
Area␣
3813␣
```

The functions:

As we did last week, you will be required to implement 2 specific functions to complete this lab successfully. Each is described below:

readData: The **readData** function will read the input file and store the four corner point values (passed as reference parameters). The function should open the input file, read from it and close it at the completion of reading the input. Thus, the stream or the stream's name should not be passed to the function. The function does not **return** a value. The four formal parameters may occur in any order of your choosing. A prototype for this function is:

```
void readData (int &, int &, int &, int &);
```

calculateArea: The **calculateArea** function is passed (by value) the corner point locations (four integers), calculates the area of the rectangle defined by the points, and returns the area calculated as an integer. A prototype for this function is:

```
int calculateArea (int, int, int, int);
```

NOTE: 100 points from this lab comes from the Curator solution, though you may be docked **100 points** for failure to solve the problem using the required functions. **20 points** will be deducted from your score if you do not adhere to the functions' interfaces (return value, name, and parameter types and passing mechanisms). The order of the formal parameters for the functions is not specified and may vary.

Teaching assistant initials:

(more on back)

Miscellaneous links:

Course web site: <http://courses.cs.vt.edu/~cs1044/fall02/mcquain>

CSI Sandbox: <http://sandbox.cs.vt.edu/~sandbox>

Curator: <http://www.cs.vt.edu/curator>

For those using subsets in conjunction with the *CSI Sandbox* program, please use the subset entitled “Project 3/Labs 8-9” during this lab!

D.9 Laboratory #9

Fall 2002 CS 1044 Lab

WEEK 9

Name: _____

Grade: _____

Part 1: Thinking about arrays

(100 points from Curator grading)

Next week, we'll be dealing with arrays in our programs. However, there are a few key concepts we would like you to start thinking about in preparation for working with arrays. The program this week gets you to consider the bounds issues that arise when working with arrays.

Input:

Assume you are given the following input file and mapping to its values

Input File (array.in)	Mapping of input file
N	U
Z	N - the number of elements in the array
value1	U - the number of "used" values in the array
value2	Z - the number of index values to process
.	
.	
valueZ	
valueZ+1	
valueZ+2	

The input file contains information which results in the description of an array to which you will simulate access in your program. The array contains "N" elements; the first "U" of these are "used" (and are considered to be holding a value). The remaining array cells are "unused". Element indices less than 0 are considered to be "invalid", and indices greater than the largest valid cell index are considered to be "out-of-bounds".

Following the values "N" and "U", the value "Z" indicates the number of remaining lines of the input file to process. You are guaranteed to receive an additional "Z" lines of input, and quite possibly may receive additional ones as well (which are to be ignored).

Each line that remains in the input file is an index which is to be tested for validity. Depending on the value, you will output the index's status ("invalid index", "unused cell", "used cell", or "index out-of-bounds").

Note: At no time are you to declare, initialize, or use an array in this program. Thus, no array values are provided as part of the input file!

Output:

The output file created by this program (**array.out**) will contain Z+1 lines of output. The first line of output will be the value of "N" and "U" separated by a single tab character. The next "Z" lines of output will be of the form: "<value>: <status>" where <value> is the numeric index you are attempting to validate, and <status> is the status of the index (see above). There is a single space following the colon on the output line.

Example:

Input file (array.in)	Output file (array.out)
10	10
5	5
6	3: used cell
3	8: unused cell
8	-1: invalid index
-1	2 used cell
2	12: index out-of-bounds
12	7: unused cell
7	
-1	
1	

Function requirements:

You are required to declare, define, and call the following function:

```
string classifyIndex (int sizeOfArray, int numUsed, int indexToClassify);
```

This function accepts three parameters: the size of the array, the number of used cells, and the index number to classify. The function returns a string value which is one of the four status values an index can have.

NOTE: 100 points from this lab comes from the Curator solution, though you may be docked **100 points** for failure to solve the problem using function(s). **20 points** will be deducted from your score if you do not adhere to the function('s) interface(s) (return value, name, and parameter types and passing mechanisms). The order of the formal parameter(s) for the function may vary.

Teaching assistant initials:

You have a maximum of 5 submissions to the Curator for this solution.

Miscellaneous links:

Course web site: <http://courses.cs.vt.edu/~cs1044/fall02/mcquain>

CS1 Sandbox: <http://sandbox.cs.vt.edu/~sandbox>

Curator: <http://www.cs.vt.edu/curator>

For those using subsets in conjunction with the *CSI Sandbox* program, please use the subset entitled “Project 3/Labs 8-9” during this lab!

D.10 Laboratory #10

Fall 2002 CS 1044 Lab

WEEK 10

Name: _____

Grade: _____

Part 1: Arrays

(100 points from Curator grading)

Project Goal:

In this lab, you will declare an array of characters, which can accommodate up to 15 values. Through the input file, you will successively set values to particular locations in the array, and print the contents of specified locations to an output file.

Input:

Input to this program will be in the input file (**array.in**). The input file will list a series of commands (at least 5 but the number is random). You should process each line of the input file and follow the command on each line. Commands can be listed in any order or sequence and are of the following form:

Command	Parameter(s)	Description
show	idx	Prints the value of the cell at the specified location (idx) in the form "Location (idx)'s value is '(val)". If the location was not initialized, or has not had a value assigned to it, you should print the message "Uninitialized location". If the location is invalid (negative) or out of bounds (> the last position in the array), you should print the message "Action invalid" to the output file.
set	idx val	Sets the value of the array location (idx) to the specified value (val) and prints the message "Location (idx) set to '(val)". The index may be invalid (negative) or out of bounds (> the last position in the array). If this is the case, you should print the message "Action invalid" to the output file.

For example, an input file may contain the following commands:

```

Input File
set 0 d
set 2 p
set 4 r
set 1 I
show 0
show 2
set -1 x
set 16 j
show 4
show -1

```

Other details:

1. If the command is not "set" or "show" (note the capitalization), the line should be ignored and the string "Invalid command" should be printed to the output file.
2. All input values are separated by a tab.
3. You can assume that you will always receive a single character in the range of a-z. The character provided will not be surrounded by single quotes.
4. You should initialize the cells of the array to a special character (e.g. "**") so that you know a value has not yet been placed in the cell.
5. You have a maximum of 5 submissions to the Curator for this solution.
6. A set command may overwrite a value in a location that was previously 'set'.

Output:

The output file created by this program should be named **array.out**. For each line of input (described above) there should be a corresponding line in the output file.

Example:

Input file (array.in)	Output file (array.out)
set 0 d	Location 0 set to d
set 2 p	Location 2 set to p
set 4 r	Location 4 set to r
set 1 I	Location 1 set to I
show 0	Location 0's value is d
show 2	Location 2's value is p
set -1 x	Action invalid
set 16 j	Action invalid
show 3	Uninitialized location
show -1	Action invalid
show 17	Action invalid

Miscellaneous links:Course web site: <http://courses.cs.vt.edu/~cs1044/fall02/mcquain>Curator: <http://www.cs.vt.edu/curator>

D.11 Laboratory #11

Fall 2002 CS 1044 Lab

WEEK 11

Name: _____

Grade: _____

Part 1: Structs

(100 points from Curator grading)

Project Goal:

Ready for structs? Good, you should be. Rather than putting related data in a wide variety of variable locations, it's time to start organizing our data in structures designed to encapsulate our data. In this lab, you'll be reading in a list of 5 movies, placing each in a struct variable, modifying the contents of the struct and then printing the results back out in a particular format.

The Structure:

Create a struct in your program that contains only the following data types. (Identifier names are up to you – the ones below are a suggestion.)

```
string movieName; // holds the name of the movie
int releaseYear; // holds the value of the year the movie was first released
string movieRating; // holds the MPAA rating the movie received upon release
string leadActor; // holds the name of the principal actor/actress in the film
```

Input:

There will be 5 films to read and process in the input file. The input file will be named (**movies.info**). Each film's listing in the input file will take on the following form:

1. The first line will contain the label "Title: " (including the space after the colon), followed by the name of the movie, a single tab, and the year of the release of the film. The release year's value will be represented by two integer digits surrounded by parentheses.
2. The second line will contain the label "Rating: " (including the space after the colon), followed by the film's rating. Remember that the rating can be G, PG, PG-13, R, etc....
3. The third line will contain the name of the lead actor/actress in the film.
4. The fourth line will be a blank line.

Given this format for each movie (4 lines of input per movie), and 5 movies in the input file, there will always be 20 lines in each input file.

Output:

Each film will be output in the format specified below to a file named (**movies.out**).

1. The first line will contain the name of the movie.
2. The second line of output will be the integer value of the release year of the movie in the form of XXXX (four integer digits).
3. The third line of the output file will contain the movie's rating. The fourth line of output will contain the actor's name.
4. The final line of output for each movie will be a line of 10 consecutive equals signs (=).

An example of the output is shown below.

Other details:

1. **Do not use an array of structs for this lab, you'll be doing that next week.**
2. The year value read from the input file is 2 digits long. However, you must prepend the value with either 19 or 20 depending on the value read to correctly generate the output. You can assume that values less than 10 are associated with the year 2000, and values greater than or equal to 10 are associated with 1900. Thus, the value 02 becomes 2002, and 72 becomes 1972.
3. You should create and call no less than 2 user-defined functions in solving this problem, **both of them should have a formal parameter that passes a structure object (the movie structure) by reference. One of them must be used to modify the year value in the structure (see #2 above).**
4. You have a maximum of 5 submissions to the curator for your solution.
5. 100 points of your grade comes from the curator. The remaining 10 points are for solving the problem with structs (5 points) and for creating and calling 2 user-defined functions (5 points).

Example:**Input file (movies.info)**

```
Title: The Matrix      (99)
Rating: R
Actor: Keanu Reeves

Title: Memento (00)
Rating: R
Actor: Guy Pearce

Title: The Silence of the Lambs      (91)
Rating: R
Actor: Jodie Foster

Title: The Usual Suspects      (95)
Rating: R
Actor: Gabriel Byrne

Title: Citizen Kane      (41)
Rating: PG
Actor: Joseph Cotten
```

Output file (movies.out)

```
The Matrix
1999
R
Keanu Reeves
=====
Memento
2000
R
Guy Pearce
=====
The Silence of the Lambs
1991
R
Jodie Foster
=====
The Usual Suspects
1995
R
Gabriel Byrne
=====
Citizen Kane
1941
PG
Joseph Cotten
=====
```

Miscellaneous links:Course web site: <http://courses.cs.vt.edu/~cs1044/fall02/mcquain>Curator: <http://www.cs.vt.edu/curator>

D.12 Laboratory #12

Fall 2002 CS 1044 Lab

WEEK 12

Name: _____

Grade: _____

Part 1: Structs Part II

(100 points from Curator grading)

Project Goal:

This week, we're going to build on last week's lab by introducing array of structures into the mix. Additionally, you will be printing only the movies requested by the input file.

The Structure:

Recall the struct definition from last week:

```
string movieName: // holds the name of the movie
int releaseYear: // holds the value of the year the movie was first released
string movieRating: // holds the MPAA rating the movie received upon release
string leadActor: // holds the name of the principal actor/actress in the film
```

Input:

The input file will be named (**movies.info**) and its first line will contain the number of movies to read and store in an array of structs (of the form defined above). You will read anywhere from 1 to 15 movies from the input file, but you will never read more than 15 movies.

The input file will be of the following form:

- The first line of the input file will be a single integer (N) indicating the number of movies to process.
- The next (N*4) lines of the input file will be in the following form, each group of 4 lines describing a single movie.
 1. The first line will contain the label "Title: " (including the space after the colon), followed by the name of the movie, a single tab, and the year of the release of the film. Two integer digits surrounded by parentheses will represent the release year's value.
 2. The second line will contain the label "Rating: " (including the space after the colon), followed by the film's rating. Remember that the rating can be G, PG, PG-13, R, etc....
 3. The third line will contain the name of the lead actor/actress in the film, preceded by the label "Actor: " (including the space after the colon).
 4. The fourth line will be blank.
- Finally, a series of "print: <name>" statements will occur. For each name listed by one of these statements, you should print the information about that movie (in the format described below). The value of <name> will be a movie title, an actor's name, or an erroneous value. If <name> is blank, or is not located as a movie or actor's name, nothing should be printed and the line should be ignored.

Output:

Each film output by this program will be in the format specified below to a file named **movies.out**.

1. The first line will contain the name of the movie.
2. The second line of output will be the integer value of the release year of the movie in the form of XXXX (four integer digits).
3. The third line of the output file will contain the movie's rating. The fourth line of output will contain the actor's name.
4. The final line of output for each movie will be a line of 10 consecutive equals signs (=).

An example of the output is shown below.

Other details:

1. You should create and call no less than 3 user-defined functions in solving this problem. They must conform to the following conventions:
 - a. At least one should take a struct variable as a parameter (not an array),
 - b. one should take a struct array as a parameter, and
 - c. one should return a struct via a **return** statement.
2. The year value read from the input file is 2 digits long. However, you must prepend the value with either 19 or 20 depending on the value read to correctly generate the output. You can assume that values less than 10 are associated with the year 2000, and values greater than or equal to 10 are associated with 1900. Thus, the value 02 becomes 2002, and 72 becomes 1972.
3. 100 points of your grade comes from the curator.

Example:**Input file (movies.info)**

```

11
Title: Citizen Kane (41)
Rating: PG
Actor: Joseph Cotten

Title: Harry Potter and the Chamber of Secrets (02)
Rating: PG
Actor: Daniel Radcliffe

Title: Alien (79)
Rating: R
Actor: Sigourney Weaver

Title: Clerks (00)
Rating: R
Actor: Kevin Smith

Title: A Few Good Men (92)
Rating: R
Actor: Kevin Bacon

Title: American Beauty (99)
Rating: R
Actor: Kevin Spacey

Title: The Matrix (99)
Rating: R
Actor: Keanu Reeves

Title: The Usual Suspects (95)
Rating: R
Actor: Gabriel Byrne

Title: I Spy (02)
Rating: PG-13
Actor: Eddie Murphy

Title: Some Like It Hot (59)
Rating: PG
Actor: Marilyn Monroe

Title: When Harry Met Sally (89)
Rating: R
Actor: Meg Ryan

Print: Alien
Print: Peter DePasquale
Print: American Beauty
Print: Nick Nolte
Print: The Usual Suspects

```

Output file (movies.out)

```

Alien
1979
R
Sigourney Weaver
*****
American Beauty
1999
R
Kevin Spacey
*****
The Usual Suspects
1995
R
Gabriel Byrne
*****

```

NOTE: 100 points from this lab comes from the Curator solution, though you may be docked **100 points** for failure to solve the problem using and array of structs. **10 points** will be deducted from your score for each function specification you fail to meet.

Teaching assistant initials:

You have a maximum of 5 submissions to the Curator for this solution.

Miscellaneous links:

Course web site: <http://courses.cs.vt.edu/~cs1044/fall02/mcquain>

Curator: <http://www.cs.vt.edu/curator>

D.13 Laboratory #13

Fall 2002 CS 1044 Lab

WEEK 13

Name:

Grade:

Part 1: Structs Part II

(100 points from Curator grading)

Project Goal:

This week, we're going to build on last week's lab by sorting the contents of the array of structures.

The Structure:

Recall the struct definition from last week:

```
string movieName; // holds the name of the movie
int releaseYear; // holds the value of the year the movie was first released
string movieRating; // holds the MPAA rating the movie received upon release
string leadActor; // holds the name of the principal actor/actress in the film
```

Input:

The input file will be named (**movies.info**) and its first line will contain the number of movies to read and store in an array of structs (of the form defined above). You will read anywhere from 1 to 15 movies from the input file, but you will never read more than 15 movies.

The input file will be of the following form:

1. The first line of the input file will be a single integer (N) indicating the number of movies to process.
2. The next (N*4) lines of the input file will be in the following form, each group of 4 lines describing a single movie.
 - a) The first line will contain the label "Title: " (including the space after the colon), followed by the name of the movie, a single tab, and the year of the release of the film. Two integer digits surrounded by parentheses will represent the release year's value.
 - b) The second line will contain the label "Rating: " (including the space after the colon), followed by the film's rating. Remember that the rating can be G, PG, PG-13, R, etc....
 - c) The third line will contain the name of the lead actor/actress in the film, preceded by the label "Actor: " (including the space after the colon).
 - d) The fourth line will be blank.

Output:

The output produced by this program will be a listing of each film in the array (each film in the input file) sorted by the title of the movie, in ascending alphabetical order. Each film output by this program will be in the format specified below to a file named **movies.out**.

1. The first line will contain the name of the movie.
2. The second line of output will be the integer value of the release year of the movie in the form of XXXX (four integer digits).
3. The third line of the output file will contain the movie's rating. The fourth line of output will contain the actor's name.
4. The final line of output for each movie will be a line of 10 consecutive equals signs (=).

Other details:

1. It is strongly suggested that you reuse your code from last week's lab. It will make the work go much, much faster.
2. The year value read from the input file is 2 digits long. However, you must prepend the value with either 19 or 20 depending on the value read to correctly generate the output. You can assume that values less than 10 are associated with the year 2000, and values greater than or equal to 10 are associated with 1900. Thus, the value 02 becomes 2002, and 72 becomes 1972.
3. 100 points of your grade comes from the Curator.
4. Refer to the attached page from your course note pack, which shows a complete code implementation sorting the contents of an array using a Bubble sort algorithm. You are free to implement any sorting algorithm you wish, but you are also welcome to adapt the Bubble sort algorithm to solve this problem.

Example:**Input file (movies.info)**

```

7
Title: A Few Good Men      (92)
Rating: R
Actor: Kevin Bacon

Title: American Beauty    (99)
Rating: R
Actor: Kevin Spacey

Title: The Matrix          (99)
Rating: R
Actor: Keanu Reeves

Title: The Usual Suspects  (95)
Rating: R
Actor: Gabriel Byrne

Title: I Spy               (02)
Rating: PG-13
Actor: Eddie Murphy

Title: Some Like It Hot    (59)
Rating: PG
Actor: Marilyn Monroe

Title: When Harry Met Sally (89)
Rating: R
Actor: Meg Ryan

```

Output file (movies.out)

```

A Few Good Men
1992
R
Kevin Bacon
*****
American Beauty
1999
R
Kevin Spacey
*****
I Spy
2002
PG-13
Eddie Murphy
*****
Some Like It Hot
1959
PG
Marilyn Monroe
*****
The Matrix
1999
R
Keanu Reeves
*****
The Usual Suspects
1995
R
Gabriel Byrne
*****
When Harry Met Sally
1989
R
Meg Ryan
*****

```

NOTE: 100 points from this lab comes from the Curator solution, though you may be docked **100 points** for failure to solve the problem using and array of structs.

You have a maximum of 5 submissions to the Curator for this solution.

Miscellaneous links:

Course web site: <http://courses.cs.vt.edu/~cs1044/fall02/mcquain>

Curator: <http://www.cs.vt.edu/curator>

This page intentionally left blank.

Appendix E

Out-of-Laboratory Project Specifications

E.1 Project #1

CS 1044 Out-of-Lab Project 1

DRAFT

Fall 2002

Putting the basics together:

Tracking Value of a Stock

It's finally time to write a complete program. This project will use many of the C++ features that were illustrated in the source code you were given for the first and second programming assignments. The resulting program will read simple data for a single stock and produce a report. This will require reading a data from an input file, doing simple calculations, and writing a simple output file. In addition, your program will have to use a C++ selection mechanism to make decisions regarding several possible charges.

Sample input data:

Here is a sample input file, named `StockData.txt`, for the program. The first line specifies the name of the customer, and the second specifies the customer's account number. The third line specifies the report date. The remaining lines specify the name of the stock, number of shares the customer holds, and the initial and final prices for the stock.

Each data value is preceded by a descriptive label that ends with a vertical bar character '| '.

Customer Name	John Q Public
Account #	432-10023-4311-89
Report Date	08/05/2002
Stock Name	MSFT
Shares held	650
Opening Price	57.32
High Price	58.14
Low Price	55.03
Closing Price	55.17

The input values are guaranteed to be syntactically and logically correct. The customer name, account number and date are just string data. For formatting purposes, you may assume that the customer name will be no more than 40 characters long, and the account number and date will be no longer than the ones shown above.

Calculations:

For the given stock, you will calculate the change in total value of all the shares the customer holds, and the percentage change in the value of a single share of the stock.

The percentage change is the change in value as a percentage of the initial stock price.

For our purposes, the change in total value and percentage change will always be reported as nonnegative values.

There is one important data representation issue in this project. As you will soon learn, computer hardware almost never stores decimal values exactly, and so any calculations done with decimal values are likely to be slightly wrong. To avoid that, your program is required to store all monetary amounts internally using integer variables. You may either store dollars and cents separately, or store a total amount in cents. Failure to do this will result in a deduction when the TAs evaluate your implementation, and also may result in a deduction when the Curator scores your submission.

Sample output:

Here is a sample output file, named `Summary.txt`, which corresponds to the input data given above:

```

Programmer: William D McQuain
CS 1044 Simple Stock Tracker

Customer:      John Q Public   432-10023-4311-89
Date:         08/05/2002
Stock        Open    Close    Loss    % Change
MSFT         57.32   55.17   1397.50  0.038

```

The first two lines just identify the programmer and project. Next there is a blank line. The next line specifies the customer name and account number, and the next specifies the report date. The next line gives the headers for the stock report. There is one small issue here. You must print the label "Gain" if the stock price increased and "Loss" if it decreased. (For this project only, the stock price is guaranteed to change.)

The final line gives the stock name, initial and final prices, total gain or loss, and the percentage change.

All monetary amounts are to be reported to two decimal places; the percentage change is to be reported to three decimal places, as shown above.

If you have read the description of how the Curator scores your program in the *Student Guide*, you know that is important that you use the same spelling and capitalization for all the labels shown above. The horizontal spacing does not effect scoring unless you combine things that should be separate or separate things that should be combined. You are free to experiment with the horizontal layout but you should try to align the columns neatly.

Suggested implementation plan:

You should design your solution piece by piece. Begin by identifying the major tasks that have to be done, and then add detail for each of those tasks. Your implementation should be developed in the same manner. Here's a suggested order of implementation.

First, implement the input code to read the customer name, account number and date, and write the output code to print them back out in the specified format. This will require declaring the appropriate stream variables and setting up the file streams, and declaring and using variables to store the input values. Test this code and make sure it produces correct results. Once you know this part works you should never have to worry about it again.

Second, implement the input code to read the stock name and other stock data. Add code to just echo the values you read in, to be sure you're reading them correctly. Note that two pieces of given information are not used in any calculations.

Then add the code to calculate the unit and total change in the stock value and to print them. Test this code. Add the calculation of the percentage change in the total value. Finally, test all of this code now to be sure it works.

By implementing the program feature by feature you let yourself concentrate on solving one small part of the problem at a time. You also should only have to test one part of it at a time as well.

Evaluation:

Everything that you have been told about testing in class applies here. Do not waste submissions to the Curator in testing your program! There is no point in submitting your program until you have verified that it produces correct results on the sample data files that are provided. If you waste all of your submissions because you have not tested your program adequately then you will receive a low score on this assignment. You will not be given extra submissions.

Your submitted program will be assigned a score, out of 100, based upon the runtime testing performed by the Curator System. We will also be evaluating your submission of this program for documentation style and a few good coding practices. This will result in a deduction (ideally zero) that will be applied to your score from the Curator to yield your final score for this project.

Read the *Programming Standards* page on the CS 1044 website for general guidelines. You should comment your code in the same manner as the code given for the first two programming assignments. In particular:

- You should have a header comment identifying yourself, and describing what the program does.
- Every constant and variable you declare should have a comment explaining its logical significance in the program.
- Every major block of code should have a comment describing its purpose.
- Adopt a consistent indentation style and stick to it.

Your implementation must also meet the following requirements:

- Choose descriptive identifiers when you declare a variable or constant. Avoid choosing identifiers that are entirely lower-case.
- Use named constants instead of literal constants when the constant has logical significance.
- Use C++ streams for input and output, not C-style constructs.
- Use C++ string variables to hold character data, not C-style character pointers or arrays.
- Note: you are explicitly forbidden to write any user-defined functions for this program. This will make the program somewhat repetitive, and physically longer than the alternative. To some extent, that's the reason for this restriction.

Understand that the list of requirements here is not a complete repetition of the *Programming Standards* page on the course website. It is possible that requirements listed there will be applied, even if they are not listed here.

Submitting your program:

You will submit this assignment to the Curator System (read the *Student Guide*), and it will be graded automatically. Instructions for submitting, and a description of how the grading is done, are contained in the *Student Guide*.

You will be allowed up to five submissions for this assignment. Use them wisely. Test your program thoroughly before submitting it. Make sure that your program produces correct results for every sample input file posted on the course website. If you do not get a perfect score, analyze the problem carefully and test your fix with the input file returned as part of the Curator e-mail message, before submitting again. The highest score you achieve will be counted.

The *Student Guide* and submission link can be found at: <http://www.cs.vt.edu/curator/>

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
// On my honor:
//
// - I have not discussed the C++ language code in my program with
//   anyone other than my instructor or the teaching assistants
//   assigned to this course.
//
// - I have not used C++ language code obtained from another student,
//   or any other unauthorized source, either modified or unmodified.
//
// - If any C++ language code or documentation used in my program
//   was obtained from another source, such as a text book or course
//   notes, that has been clearly noted with a proper citation in
//   the comments of my program.
//
// - I have not designed this program in such a way as to defeat or
//   interfere with the normal operation of the Curator System.
//
// <Student Name>
```

Failure to include this pledge in a submission is a violation of the Honor Code.

E.2 Project #2

Note that the header of the project's specification pages is incorrect (labeled as Project #4).

CS 1044 Program 4	DRAFT	Fall 2002
--------------------------	--------------	------------------

Input Failure Control and Decisions: **Simple Stock Portfolio Tracker**

For this project you will modify and extend your implementation for Project 1 to employ an input-failure/sentinel loop and more sophisticated calculations. The input and output files will be radically different.

Sample input data:

Here is a sample input file, named `StockData.txt`, for the program. The first line specifies the name of the customer, and the second specifies the customer's account number. The third line specifies the report date. Each of the remaining lines specifies the name of a particular stock, the number of shares the customer holds, and the initial, high, low, and final prices for the stock. These values are separated by single tab characters.

Customer Name	Terry Pratchett				
Account #	520-103205-2817-56				
Report Date	02/21/2002				
IBM	1856	67.90	72.49	58.23	62.42
AT&T	7689	9.31	14.14	5.21	5.21
Lucent	8020	1.60	4.29	0.01	0.59
FannieMae	821	71.51	81.32	61.80	67.15
PepsiCo	9497	40.52	42.09	39.00	42.02
End of data.	0	0.00	0.00	0.00	0.00

The final line of stock data may (or may not) be followed by a sentinel line as shown above. There is no limit on the number of lines of stock data that may be supplied; your program must terminate properly when the sentinel value is detected, or when the input stream fails, whichever occurs first. If there is a sentinel line, the period will be followed by a tab character to be consistent with the preceding lines of stock data.

The input values are guaranteed to be syntactically and logically correct. The customer name, account number and date are just string data. For formatting purposes, you may assume that the customer name will be no more than 40 characters long, and the account number and date will be no longer than the ones shown above.

Calculations:

For each of the given stocks, you will calculate the change in total value of all the shares the customer holds, and the percentage change in the value of a single share of the stock.

As before, the percentage change is the change in value as a percentage of the initial stock price. One change from the previous project is that the total change and percentage change are now reported as signed values.

We add a summary report, including the total initial and final value for all the shares of all the stocks, the overall change, and the overall percentage change.

The comments on representing monetary values given for Project 1 still apply.

Sample output:

The first two lines of the output file just identify the programmer and project. Next there is a blank line. The next line specifies the customer name and account number, and the next specifies the report date. The next line gives the headers for the stock report table.

Each line of the stock report table gives the number of shares, initial and final price, total change and percentage change for one stock. The table is bounded above and below by a single line of delimiters. The first summary line gives the overall summary data for the entire stock portfolio. Next there is a blank line, followed by a report on how many stocks increased in value and how many decreased in value. (Stocks with unchanged value are not counted.)

Page 1 of 4

CS 1044 Program 4

DRAFT

Fall 2002

All monetary amounts are to be reported to two decimal places; the percentage changes are to be reported to three decimal places, as shown above.

Here is a sample output file, named `Summary.txt`, which corresponds to the input data given above:

```

Programmer: William D McQuain
CS 1044 Simple Portfolio Tracker

Customer: Terry Pratchett 520-103205-2817-56
Date: 02/21/2002
Stock      Shares      Open      Close      Change      % Change
-----
IBM        1856        67.90     62.42     -10170.88   -0.081
AT&T       7689         9.31      5.21     -31524.90   -0.440
Lucent     8020         1.60      0.59     -8100.20    -0.631
FannieMae 821          71.51     67.15    -3579.56    -0.061
PepsiCo    9497         40.52     42.02    14245.50    0.037
-----
Summary                    653967.14  614837.10  -39130.04   -0.060

Winners: 1
Losers: 4

```

If you have read the description of how the Curator scores your program in the *Student Guide*, you know that is important that you use the same spelling and capitalization for all the labels shown above. The horizontal spacing does not effect scoring unless you combine things that should be separate or separate things that should be combined. You are free to experiment with the horizontal layout but you should try to align the columns neatly.

Suggested implementation plan:

You should design your solution piece by piece. Begin by identifying the major tasks that have to be done, and then add detail for each of those tasks. Your implementation should be developed in the same manner. Here's a suggested order of implementation.

First, implement the input code to read the customer name, account number and date, and write the output code to print them back out in the specified format. This will require declaring the appropriate stream variables and setting up the file streams, and declaring and using variables to store the input values. Test this code and make sure it produces correct results. Once you know this part works you should never have to worry about it again.

Second, implement the input code to read the stock name and other stock data, assuming the sentinel line is present. Add code to just echo the values you read in, to be sure you're reading them correctly, and stopping at the right place. Next, modify this code to also stop correctly on an input failure, and test that (both with and without a sentinel line).

Then add the code to calculate the unit and total change in the stock value and to print them. Test this code. Add the calculation of the percentage change in the total value. Finally, test all of this code now to be sure it works.

By implementing the program feature by feature you let yourself concentrate on solving one small part of the problem at a time. You also should only have to test one part of it at a time as well.

Evaluation:

Everything that you have been told about testing in class applies here. Do not waste submissions to the Curator in testing your program! There is no point in submitting your program until you have verified that it produces correct results on the sample data files that are provided. If you waste all of your submissions because you have not tested your program adequately then you will receive a low score on this assignment. You will not be given extra submissions.

Your submitted program will be assigned a score, out of 100, based upon the runtime testing performed by the Curator System. We will also be evaluating your submission of this program for documentation style and a few good coding practices. This will result in a deduction (ideally zero) that will be applied to your score from the Curator to yield your final score for this project.

Read the *Programming Standards* page on the CS 1044 website for general guidelines. You should comment your code in the same manner as the code given for the first two programming assignments. In particular:

- You should have a header comment identifying yourself, and describing what the program does.
- Every constant and variable you declare should have a comment explaining its logical significance in the program.
- Every major block of code should have a comment describing its purpose.
- Adopt a consistent indentation style and stick to it.

Your implementation must also meet the following requirements:

- Choose descriptive identifiers when you declare a variable or constant. Avoid choosing identifiers that are entirely lower-case.
- Use named constants instead of literal constants when the constant has logical significance.
- Use C++ streams for input and output, not C-style constructs.
- Use C++ string variables to hold character data, not C-style character pointers or arrays.
- Note: you are explicitly forbidden to write any user-defined functions for this program. This will make the program somewhat repetitive, and physically longer than the alternative. To some extent, that's the reason for this restriction.

Understand that the list of requirements here is not a complete repetition of the *Programming Standards* page on the course website. It is possible that requirements listed there will be applied, even if they are not listed here.

Submitting your program:

You will submit this assignment to the Curator System (read the *Student Guide*), and it will be graded automatically. Instructions for submitting, and a description of how the grading is done, are contained in the *Student Guide*.

You will be allowed up to five submissions for this assignment. Use them wisely. Test your program thoroughly before submitting it. Make sure that your program produces correct results for every sample input file posted on the course website. If you do not get a perfect score, analyze the problem carefully and test your fix with the input file returned as part of the Curator e-mail message, before submitting again. The highest score you achieve will be counted.

The *Student Guide* and submission link can be found at: <http://www.cs.vt.edu/curator/>

CS 1044 Program 4

DRAFT

Fall 2002

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
// On my honor :  
//  
// - I have not discussed the C++ language code in my program with  
// anyone other than my instructor or the teaching assistants  
// assigned to this course.  
//  
// - I have not used C++ language code obtained from another student,  
// or any other unauthorized source, either modified or unmodified.  
//  
// - If any C++ language code or documentation used in my program  
// was obtained from another source, such as a text book or course  
// notes, that has been clearly noted with a proper citation in  
// the comments of my program.  
//  
// - I have not designed this program in such a way as to defeat or  
// interfere with the normal operation of the Curator System.  
//  
// <Student Name>
```

Failure to include this pledge in a submission is a violation of the Honor Code.

E.3 Project #3

CS 1044 Program 3

Fall 2002

Procedural Decomposition:

Simple Stock Portfolio Tracker

For this project you will modify and extend your implementation for Project 2 to accommodate a design based upon a good procedural decomposition of the assigned problem. The input file will have precisely the same format as for the previous project. The output file will be slightly modified to reflect some additional requirements.

Sample input data:

Here is a sample input file, named `StockData.txt`, for the program. The first line specifies the name of the customer, and the second specifies the customer's account number. The third line specifies the report date. Each of the remaining lines specify the name of a particular stock, the number of shares the customer holds, and the initial and final prices for the stock. These values are separated by single tab characters.

Customer Name	John Q Public				
Account #	432-10023-4311-89				
Report Date	08/05/2002				
MSFT	650	57.32	58.14	55.03	55.17
IBM	100	53.02	54.27	53.02	54.00
APPL	200	17.45	17.45	13.09	13.09

There is no limit on the number of lines of stock data that may be supplied; your program must terminate properly when the input stream fails. There will be no sentinel line. The input values are guaranteed to be syntactically and logically correct. The customer name, account number and date are just string data. For formatting purposes, you may assume that the customer name will be no more than 40 characters long, and the account number and date will be no longer than the ones shown above.

Calculations:

For each of the given stocks, you will still calculate all of the values specified for the previous project. In addition, you will also calculate a measure of the volatility of each stock's performance. For the purposes of this assignment, we will measure the volatility as the total range of the stock's price movement as a percentage of the opening price of the stock. More specifically, the volatility involves the sum of the absolute values of three quantities: the difference between the high and low prices, the difference between the high price and the smaller of the initial and closing prices, and the difference between the low price and the larger of the initial and closing prices. In the event the initial price is zero, we will arbitrarily define the volatility to also be zero; however that situation will not arise in actual data.

The comments on representing monetary values given for Project 1 still apply.

Sample output:

Here is a sample output file, named `Summary.txt`, which corresponds to the input data given above:

Programmer: William D McQuain						
CS 1044 Simple Portfolio Tracker						
Customer:	John Q Public	432-10023-4311-89				
Date:	08/05/2002					
Stock	Shares	Open	Close	Change	% Change	Volatility
MSFT	650	57.32	55.17	-1397.50	-0.038	0.15
IBM	100	53.02	54.00	98.00	0.018	0.07
APPL	200	17.45	13.09	-872.00	-0.250	0.75
Summary		46050.00	43878.50	-2171.50	-0.047	
Winners:	1					
Losers:	2					

CS 1044 Program 3**Fall 2002**

The format of the output file is identical to that for Project 2, except that there is an additional column showing the volatility of each stock; you must report the volatility to two decimal places.

If you have read the description of how the Curator scores your program in the *Student Guide*, you know that is important that you use the same spelling and capitalization for all the labels shown above. The horizontal spacing does not effect scoring unless you combine things that should be separate or separate things that should be combined. You are free to experiment with the horizontal layout but you should try to align the columns neatly.

Procedural decomposition:

As always, you should practice incremental development. In this case, you have already done much of the low-level design work in completing the previous project. You should also be able to recycle much of the C++ source code you wrote for the previous project.

The primary new design issue is that you must take advantage of the opportunity to divide the assignment into independent procedures. Your implementation must reflect this by making good use of C++ functions. Specific minimal requirements are given in the Evaluation section that follows; be sure you follow them.

Evaluation:

Everything that you have been told about testing in class applies here. Do not waste submissions to the Curator in testing your program! There is no point in submitting your program until you have verified that it produces correct results on the sample data files that are provided. If you waste all of your submissions because you have not tested your program adequately then you will receive a low score on this assignment. You will not be given extra submissions.

Your submitted program will be assigned a score, out of 100, based upon the runtime testing performed by the Curator System. We will also be evaluating your submission of this program for documentation style and a few good coding practices. This will result in a deduction (ideally zero) that will be applied to your score from the Curator to yield your final score for this project.

Read the *Programming Standards* page on the CS 1044 website for general guidelines. You should comment your code in the same manner as the code given for the first two programming assignments. In particular:

- You should have a header comment identifying yourself, and describing what the program does.
- Every constant and variable you declare should have a comment explaining its logical significance in the program.
- Every major block of code should have a comment describing its purpose.
- Adopt a consistent indentation style and stick to it.

Your implementation must also meet the following requirements:

- Choose descriptive identifiers when you declare a variable or constant. Avoid choosing identifiers that are entirely lower-case.
- Use named constants instead of literal constants when the constant has logical significance.
- Use C++ streams for input and output, not C-style constructs.
- Use C++ string variables to hold character data, not C-style character pointers or arrays.
- Use C++ functions intelligently and appropriately. In particular, you must implement and use at least 4 functions in addition to `main()`. At least one of these functions must be `void`; at least one must return a value via a `return` statement, and at least one must use a reference parameter to return a value. It is acceptable that all function calls are from `main()` to other functions.
- Use the appropriate protocol when passing each parameter to a function. In particular, do not pass any parameter by reference unless it is being used to communicate a changed value from the called function to its caller.

Important: since the primary functionality of this program was already present in the previous project, you must meet the functional decomposition requirements given above in order to receive full credit. A program that uses only one function, `main()`, will be assigned a final score of zero. Programs that use fewer than the specified number of functions will receive substantial penalties.

CS 1044 Program 3**Fall 2002**

Understand that the list of requirements here is not a complete repetition of the *Programming Standards* page on the course website. It is possible that requirements listed there will be applied, even if they are not listed here.

Submitting your program:

You will submit this assignment to the Curator System (read the *Student Guide*), and it will be graded automatically. Instructions for submitting, and a description of how the grading is done, are contained in the *Student Guide*.

You will be allowed up to five submissions for this assignment. Use them wisely. Test your program thoroughly before submitting it. Make sure that your program produces correct results for every sample input file posted on the course website. If you do not get a perfect score, analyze the problem carefully and test your fix with the input file returned as part of the Curator e-mail message, before submitting again. The highest score you achieve will be counted.

The *Student Guide* and submission link can be found at: <http://www.cs.vt.edu/curator/>

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
// On my honor:
//
// - I have not discussed the C++ language code in my program with
//   anyone other than my instructor or the teaching assistants
//   assigned to this course.
//
// - I have not used C++ language code obtained from another student,
//   or any other unauthorized source, either modified or unmodified.
//
// - If any C++ language code or documentation used in my program
//   was obtained from another source, such as a text book or course
//   notes, that has been clearly noted with a proper citation in
//   the comments of my program.
//
// - I have not designed this program in such a way as to defeat or
//   interfere with the normal operation of the Curator System.
//
// <Student Name>
```

Failure to include this pledge in a submission is a violation of the Honor Code.

E.4 Project #4

CS 1044 Project 4

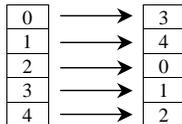
DRAFT

Fall 2002

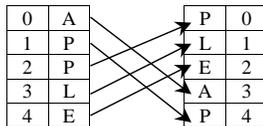
User-defined Functions and Arrays

Permutation-Based Cryptography

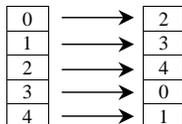
The translation of text from clear form to an encrypted form and back is a common problem in computing. One simple technique for encrypting text is based on the mathematical notion of a permutation. A permutation of the integers 0 through $N-1$ is simply a one-to-one function whose domain and range are both the set of integers $\{0, 1, 2, \dots, N-1\}$. In other words, a permutation transforms each integer in the set $\{0, 1, 2, \dots, N-1\}$ into another integer in the same set, with no two integers being transformed into the same result. For example, here is a permutation of $\{0, 1, 2, 3, 4\}$:



This permutation can be used to encrypt a sequence of five characters by moving each character from its original position to the position defined by the permutation. For example, the sequence "APPLE" would be translated into the string "PLEAP":



Note that we number positions starting at zero just as in C++ arrays — that's a hint of things to come. What about decrypting the text above? Well, each permutation has an *inverse*, another permutation that does the exact opposite of the original permutation. For the permutation given above, the inverse permutation is:



Apply the inverse permutation to the scrambled sequence "PLEAP"; you should get back the original sequence "APPLE". That's the key point about encryption/decryption using a permutation:

Apply the permutation, and then apply its inverse, and you get back where you started.

That means we can use a permutation to encrypt a sequence and then use the inverse of that permutation to decrypt the encrypted sequence and get back the original.

For this project, you will write a program that is capable of both encrypting text and decrypting text, applying a given permutation to sequences of characters taken from the original text.

Encryption: Given a permutation P and a sample of clear (unencrypted) text T_C , we can apply the permutation to the clear text to obtain encrypted text T_E . The process is described in this section.

The scheme described above assumes that the permutation is the same length as the string of characters you are working with. That's OK for individual words, but not so good for long text samples like:

```
To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
The slings and arrows of outrageous fortune,
Or to take arms against a sea of troubles,
And by opposing end them.
```

There are 199 characters there, including the newlines. While we could create a permutation of $\{0, 1, \dots, 198\}$ and use it to encrypt this text, there's a simpler approach. We can create a shorter permutation, say of $\{0, 1, \dots, 19\}$ and use it to encrypt "chunks" of 20 characters until we're done with the entire text (aside from a slight problem with the last chunk of text).

For instance, take the permutation:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
6	13	1	15	8	3	11	7	9	12	10	16	14	17	18	4	19	0	2	5

Now read the first 20 characters from the text given above and apply the permutation:

T	o		b	e	,		o	r		n	o	t		t	o		b	e	:
b		e	,	o	:	T	o	e	r	n				o	t	b	o		t

then read the next 20 characters and repeat the process:

	t	h	a	t		i	s		t	h	e		q	u	e	s	t	i	o
t	h	i		e	o		s	t		h	i	t	t		a	e	q	u	s

and again (note that the newline character at the end of the first line is treated just like any other character):

n	:	\n	w	h	e	t	h	e	r		'	t	i	s		n	o	b	l
o	\n	b	e		l	n	h	h	e		t	r	:	t	w	'	i	s	n

So the text considered so far would encrypt as follows:

```
b e,o:Toern otbo t thi eo st hitt aequso
be lnhhe tr:tW'isn
```

Continuing this way we would obtain the complete encryption:

```
b e,o:Toern otbo t thi eo st hitt aequso
be lnhhe tr:tW'isn s tuehnemt rniid onedT fe
lhsfnrigsaorusgsaowfo rtouraeaoku e et,On
f rrtotera gsanaiatms asb yod obrlsuef
t,An .p
iemogs nnep odth
```

There's just one little problem. The permutation takes 20 characters at a time and the total number of characters isn't a multiple of 20. That means that when we get down to the end we don't have enough characters in the last "chunk" of input to match up with the permutation entries.

The End Game: When we get to the end of the sample text our logic must be altered slightly. Consider the sample input text given above. There are 199 characters in that sample and the permutation takes 20 of them at a time. That means that after we've read and processed 9 chunks of text there will be 19 characters left. That doesn't match the length of our permutation, so the approach described before won't quite work. What we need is a permutation that matches the length of this last chunk. Here's a simple (and for this project, required) way to create such a permutation.

Take a permutation of $\{0, \dots, 19\}$, say the one from above:

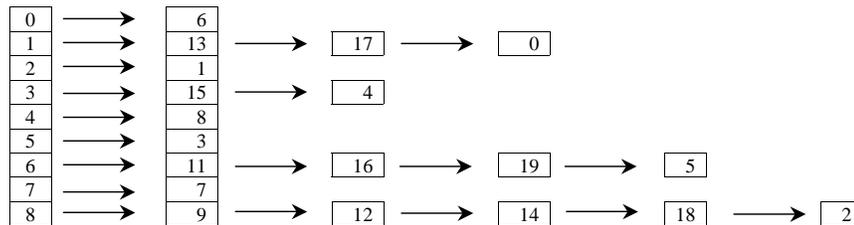
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
6	13	1	15	8	3	11	7	9	12	10	16	14	17	18	4	19	0	2	5

Let's say we need a permutation of $\{0, \dots, 8\}$ to handle 9 characters. We may construct such a permutation from the one above in the following way. Let k be in $\{0, \dots, 8\}$. Let k' be the value the original permutation maps k to. If k' is in $\{0, \dots, 8\}$ just keep that value. That takes care 0, 2, 4, 5, and 7 above. If k' is bigger than 8, we look at the value the original permutation maps k' to; call that value k'' . If k'' is in the range $\{0, \dots, 8\}$, then let the new permutation map k to k'' . If k'' is bigger than 8, continue the process by looking at the value the original permutation maps k'' to. Eventually you have to get a value that's in the range $\{0, \dots, 8\}$; when you do, let the new permutation map k to that.

Applying this idea to the permutation given above, we get:

0	1	2	3	4	5	6	7	8
6	0	1	4	8	3	5	7	2

Why? Well, using the original permutation given above, we have:



Now, once we've constructed the right size permutation we can apply it to encrypt the last chunk of the original text sample. Recall the text sample from the previous page. The last text chunk is 19 characters long (including the newline at the end), so we need a permutation of $\{0, \dots, 19\}$. Using the technique just shown above, we get the permutation:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
6	13	1	15	8	3	11	7	9	12	10	16	14	17	18	4	5	0	2

Now, the last chunk would be the text shown below, encrypted as shown:

o	p	p	o	s	i	n	g		e	n	d		t	h	e	m	.	\n
.	p	\n	i	e	m	o	g	s		n	n	e	p		o	d	t	h

Decryption: The decryption problem: given a sample of encrypted text, T_E , we wish to obtain the corresponding clear text T_C . The process depends upon one simple fact:

Given a clear text sample T_C and a permutation P , there is only one encrypted text T_E that can be obtained, if the process described above is used.

Thus, decrypting text is, mathematically speaking, the inverse of the encryption operation. Fortunately, we may solve the decryption problem in a very simple manner. Every permutation P has an inverse permutation, usually denoted by P^{-1} . If we know P^{-1} , we may decrypt an encrypted text sample T_E by applying the inverse permutation to T_E in exactly the way the original permutation P was applied to create T_E .

If we know P , we may find P^{-1} easily: if P maps i to j , then P^{-1} will map j to i . (Take another look at the discussion on the first page of this spec if that wasn't clear.)

Let's take another look at the encrypted version of that quotation from *Hamlet*:

```
b e,o:Toern otbo t thi eo st hitt aequso
be lnhhe tr:tW'isn s tuehnmT rniid onedT fe
lhsfnrigsaorusgsaowfo rtouraeaoku e et,On
f rrtotera gsanaiatms asb yod obrlsuef
t,An .p
iemogs nnep odth
```

We obtained this using the permutation:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
6	13	1	15	8	3	11	7	9	12	10	16	14	17	18	4	19	0	2	5

The inverse permutation would be:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
17	2	18	5	15	19	0	7	4	8	10	6	9	1	12	3	11	13	14	16

Apply the inverse permutation to the first chunk of the encrypted text:

b		e	,	o	:	T	o	e	r	n			o	t	b	o		t	
T	o		b	e	,		o	r		n	o	t		t	o		b	e	:

Continue in this way, applying the inverse permutation to successive chunks until all the encrypted text has been processed and you'll obtain the original text.

Aside: What makes this technique effective is that to decrypt text you have to know the permutation P that was used to encrypt the original text. Given the encrypted text above, you don't even know how long the permutation P was. Even knowing that P was of length 20, there are $20! = 20 \cdot 19 \cdot 18 \cdot 17 \cdot \dots \cdot 3 \cdot 2 \cdot 1$ possible permutations of that length. The amount of time required to try each one of those (the brute force approach) is prohibitive if you need the answer quickly. Without knowing the length of P, the amount of time for a brute force solution becomes astronomical. It can be made even worse if the permutation itself is varied from chunk to chunk of input text.

From the back of an envelope: $20! \approx 2.43 \times 10^{18}$ — if it took one one-thousandth of a second to apply a permutation and then determine whether the resulting text made sense, processing this many permutations would take about 77,146,816 years. Of course, faster hardware would help, but it would still take over 77,000 years if one million permutations could be processed per second. Not to mention it's possible that applying the wrong permutation might produce sensible, but incorrect, text.

Input:

Your program **must** read its input from a file named `CryptoData.txt` — use of another input file name will result in a score of zero. The first two lines of the input file specify the permutation to be used. You should treat the first line as a comment (ignore it) and read the values on the second line into an appropriate array — the permutation will always be of length 20.

The next section of the input file is a data header, containing five lines of labels and data describing the text sample to be processed. The first two lines of the data header are a comment and a line containing a single character specifying whether the text sample that follows is to be encrypted ('E') or decrypted ('D') using the given permutation. The next two lines are another comment and a line containing the number of characters included in the text sample, counting spaces and newlines. The last line of the data header is just a comment.

The text sample to be encrypted or decrypted follows the data header. You may assume that the text sample will contain exactly the specified number of characters. There will be one or two blank lines after the last character of the text sample (not counted as a part of the text sample).

After that, there will be a second data header section and another text sample to be processed. Each input file will contain exactly two text samples for processing, one to decrypt and one to encrypt (in random order).

The values on each line will be separated by whitespace. You may assume that all the input values will be logically correct. For instance:

```

Permutation: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
             3 9 0 1 10 2 4 15 14 5 16 11 17 12 6 18 7 8 13 19

Option: D
Number of characters: 220
ouhT rWodhga iBnmobcoee ian museotDnn,ndt
h sd,A po uoopebngoefoa bi wmon onr
Ytn wty ,el t Iilrhlate.Bemy sfr eo by
oto ardI ylrhwmwi sike.yoneh a,dI L cdfMfAaneauddd
;n m hmb art ei ichttfsrs Hio,uh!e"eo"dI ng

Option: E
Number of characters: 141
If thou didst ever hold me in thy heart
Absent thee from felicity awhile,
And in this harsh world draw thy breath in pain,
To tell my story.

```

Note that you must **not make any assumptions** about the number of lines of data in the input file; the input file will comply with the specification given here. Your program must be written so that it will detect the end of each text sample correctly.

What to Calculate:

There are almost no calculations in the numerical sense. You will write a program to read the input file and use the given permutation to encrypt or decrypt (as specified) each of the two text samples. The processed text will be written to an output file, formatted as described in the next section.

Output:

The first line of your output should include your name only; the second line should include the project title as shown in the sample file. The next line should be a blank line, followed by a delimiter line to separate the file header from the output generated when the first text sample is processed. The length and characters used in the delimiter line do not matter, but it must not be blank.

Next your output file will specify how many characters were processed and whether they were encrypted or decrypted, and then echo the permutation used, as shown. Following the header is the processed text. Do not change the capitalization of the input text, and do not add or remove any newlines within the text you are processing: if the input sample text is processed properly, you should automatically generate appropriate line breaks (newlines) when you print it. You may generate a blank line at the end of the processed text, or not, as you like.

Following that, there will be another, identical, section of output generated when the second input sample text is processed. A delimiter line and a line containing the exit message shown below should follow the end of that section.

Your program must write output data to a file named `CryptoOut.txt` — use of any other output file name will result in a score of zero. The sample output file shown below corresponds to the input data given above:

- Use `char` arrays, of the appropriate dimension, to hold the text being encrypted or decrypted. Note: using `string` variables to do this is strictly prohibited. You may use `string` variables for other purposes.
- Use at least three arrays, including at least one of type `int` and at least two of type `char`.
- You may use file-scoped function prototypes, and you may use file-scoped constants.
- You may not use file-scoped variables of any kind.
- You must make good use of user-defined functions in your design and implementation. To encourage this, the body of `main()` must contain no more than 20 executable statements and the bodies of the other functions you write must each contain no more than 40 executable statements. An executable statement is any statement **other than** a constant or variable declaration, function prototype or comment. Blank lines do not count.
- You must write at least six functions, besides `main()`. For reference, my solution uses eight: two for input, two for output, two for text translation, one to find the inverse of a permutation, and one utility function.
- The definition of `main()` must be the first function definition in your source file. The remaining function definitions should be grouped logically.
- Function parameters should be passed appropriately. Use pass-by-reference only when the called function needs to modify the parameter. Pass array parameters by constant reference (using `const`) when pass-by-reference is not needed.
- Use named constants instead of variables where appropriate — in particular for array dimensions.
- Choose your control structures appropriately.

Submitting your program:

You will submit this assignment to the Curator System (read the *Student Guide*), and it will be graded automatically. Instructions for submitting, and a description of how the grading is done, are contained in the *Student Guide*.

You will be allowed up to five submissions for this assignment. Use them wisely. Test your program thoroughly before submitting it. Make sure that your program produces correct results for every sample input file posted on the course website. If you do not get a perfect score, analyze the problem carefully and test your fix with the input file returned as part of the Curator e-mail message, before submitting again. The highest score you achieve will be counted.

The *Student Guide* and submission link can be found at: <http://www.cs.vt.edu/curator/>

Incremental Development:

You'll find that it's easier and faster to produce a working program by practicing incremental development. In other words, don't try to solve the entire problem at once. First, develop your design. When the time comes to implement your design, do it piece by piece. Here's a suggested implementation strategy for this project. As usual, verify and correct as necessary after each addition to your implementation.

- First, focus on reading a single text sample. Use the sample input given with this specification, but delete the section containing the sample to be decrypted. Implement code to read the file header (permutation and sample text description) and make sure those are read correctly. Then implement code to read the sample text in 20-character chunks — echo it without any changes to be sure you've got that right and that you're handling the last, short chunk correctly.
- Second, implement the encryption code to translate the text (you're now reading correctly) using the given permutation. This shouldn't require much modification of the earlier code since you can use the same output code here as well. Test this thoroughly. A number of samples will be posted to the website.
- Third, add the logic to determine whether you're to encrypt or decrypt, if you didn't write that earlier. Then implement the decryption code. That should be a fairly simple modification of the decryption code you wrote earlier, at least if you're thinking about the process correctly. Again, you should be use the same output code whether you're printing encrypted or decrypted text to the output file.

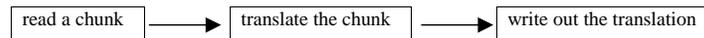
Now you have a substantially complete program. At this point, you should clean up your code, eliminating any unnecessary instructions and fine-tuning the documentation you already wrote. Check your implementation and output again to be sure that you've followed all the specifications given for this project, especially those in the Evaluation section above. At this point, you're ready to submit your solution to the Curator.

Implementation Hints:

You can store the given permutation P in an array, say $A[]$. Then $A[k] == j$ if and only if the permutation P maps k to j . Another way of saying this is that $A[k]$ is the location to which the character at position k should be moved. So, you only have to store the “second line” of P .

You can also think of the inverse permutation P^{-1} in terms of the array $A[]$: if $A[k] == j$ then P^{-1} would map j to k ; said another way, the character that’s at position j should really be at position k .

About the requirement you declare and use enumerated type. Notice that you read the chunks of the text sample the same way whether you’re going to encrypt it or decrypt it, and the same goes for writing the translated chunks to the output file. If you understand the hints above, encrypting and decrypting are (different but) very similar. The basic model of the processing here is something like:



All you have to do is select the appropriate type of translation (encryption or decryption).

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```

// On my honor:
//
// - I have not discussed the C++ language code in my program with
//   anyone other than my instructor or the teaching assistants
//   assigned to this course.
//
// - I have not used C++ language code obtained from another student,
//   or any other unauthorized source, either modified or unmodified.
//
// - If any C++ language code or documentation used in my program
//   was obtained from another source, such as a text book or course
//   notes, that has been clearly noted with a proper citation in
//   the comments of my program.
//
// - I have not designed this program in such a way as to defeat or
//   interfere with the normal operation of the Curator System.
//
// <Student Name>
  
```

Failure to include this pledge in a submission is a violation of the Honor Code.

E.5 Project #5

CS 1044 Program 5

Fall 2002

Data Encapsulation and Management:

Stock Portfolio Database

For this project you will modify and extend your implementation for Project 3 to take advantage of arrays and structures to improve the organization of the data and to support additional operations. You will also explore searching and sorting arrays of structures.

There will be two input files. The first will contain data about a list of stocks. The second will be a script file containing database queries and actions that your program must interpret and carry out.

On startup, your program will read the stock data file and organize the given data in an array of structures. To do this, you must design and use an appropriate `struct` type. Your program will then read the script file and service the queries or carry out the actions, writing descriptive output to a report file.

Database queries and actions:

Your program must be able to service the queries and actions specified below. In general, each query or action will require performing a search for a stock matching some criterion and then reporting or modifying some values for that stock.

show<tab><stock name>

Search the database for a stock with the given name. If a match is found, log the stock name, number of shares, buy price and current price, and the total value of all the held shares to the report file, along with the index at which the stock was found. If no match is found, log the error message "Not found" to the report file.

net change<tab><stock name>

Search the database for a stock with the given name. If a match is found, log the change in the total value of the held shares since the stock was bought. If no match is found, log the error message "Not found" to the report file.

buy<tab><stock name><tab><shares>

Search the database for a stock with the given name. If a match is found, increase the number of shares held by the specified amount and log the message "Bought for" followed by the total amount paid for the shares that were bought, assuming the current price was paid. If no match is found, log the error message "Not found" to the report file.

sell<tab><stock name><tab><shares>

Search the database for a stock with the given name. If a match is found, decrease the number of shares held by the specified amount and log the message "Sold for" followed by the total value of the sold shares. If no match is found, log the error message "Not found" to the report file. If the number of shares held is insufficient, log the error message "Not enough shares".

quote<tab><stock name><tab><price>

Search the database for a stock with the given name. If a match is found, change the current price to the specified amount and log the message "<stock name> updated to" followed by the new current price. If no match is found, log the error message "Not found" to the report file.

sort by<tab>[**name** | **value**]

Sort the database into ascending order, either with respect to the stock name or the total value of the held shares stocks. You must use the bubble sort algorithm when sorting by value and the selection sort algorithm when sorting by name. In either case, log the total number of swaps performed during the sort to the report file.

total value<tab>

Log the total value of all held shares of all stocks to the report file.

display<tab>

Log a list of all stock information from the database. The display must include the index number for each stock, followed by the stock name, number of shares, opening and current prices, and the total value of the shares held. See the sample log file for formatting details.

CS 1044 Program 5**Fall 2002****exit**<tab>

Stop processing the script file and log the message "Exiting script processing".

Sample input data files:

The stock data file will be named `PortfolioDB.txt`. There is no customer information. There is a two-line header, followed by a list of data lines. For each stock, you will be given the name, number of shares held, the price per share at which the stock was purchased, and the price of a share. These values are separated by single tab characters.

Stock	Shares	Bought	Current
Microsft	467	45.67	44.62
IBM	985	67.90	71.01
Lucent	413	1.60	4.23
DellCptr	811	24.32	21.44
AT&T	1017	9.31	6.12
CSX	404	33.66	41.32

The number of lines of stock data that may be supplied will vary, but there will never be data for more than 100 stocks; your program must terminate properly when the input stream fails. There will be no sentinel line. The input values are guaranteed to be syntactically and logically correct.

The script file will be named `Queries.txt`. There is no limit on the number of queries that may be supplied, but you should design your program to terminate correctly if it encounters an `exit` command or an input failure occurs.

```
show      Microsft
net change      Microsft
sell       Microsft      100
show      Microsft
total value
buy       Microsft      200
show      Microsft
show      Citi
quote    DellCptr      22.50
show      DellCptr
sort by   name
display
sort by   value
display
exit
```

A sample log file corresponding to these input files is given at the end of this specification.

Design and implementation:

Note: the comments on design are more than just advice; the comments on implementation are merely (good) advice.

The first thing you need to design is a `struct` type that will represent a single stock. It's up to you to determine what data members the type should have, but all of the data relevant to a single stock must be stored within a single `struct` variable. In order to aid in detecting errors, you should initialize the entire data array to hold recognizable, dummy values.

You should practice good procedural decomposition in your design. Reading the stock data from the file would be a good candidate for a function. Managing the reading and interpretation of the queries and actions from the script would be

CS 1044 Program 5**Fall 2002**

another good job for a function. The details of handling each particular query or action could be delegated to a particular "handler" function. Of course, some of the handlers might also use helper functions, to perform tasks such as sorting or searching. In addition, there are the usual opportunities to have functions to handle mundane details, such as writing the file header. For reference, my solution uses 21 functions in addition to `main()`.

You should take care to pass parameters intelligently. In particular, do not pass the array of stock data by reference unless the logic of the function requires it. Do not pass unnecessary parameters. Avoid implementing a function whose body is only one statement unless the function is called from more than one place.

As always, you should practice incremental development. Begin by implementing your structured type and the logic needed to read the stock data and store it in the array of structured variables. Write output code to display the full list of stock data and verify that your input logic is correct.

Next implement a function to read the commands from the script file, and identify each command. Knowing which command is given, you can then select what actions to take. Initially, just log a message that identifies the command, but does not attempt to carry it out. Once you're sure you are reading and classifying the commands correctly, add handlers for each command.

Add and test the handlers one by one; test each one thoroughly before moving on to the next one. Some handlers will be more complicated than others. The most critical operation is searching, since that is used in servicing several commands. Be sure to test your search code thoroughly! The most complex commands are probably the sorting commands. You will need two different sort functions, since two different sort algorithms are required. (Yes, I know you could combine those into a single, overly-complex, non-reusable function.)

If you do this well, you will produce a design for handling script-driven processing that you can reuse in later projects (in later courses).

Evaluation:

Everything that you have been told about testing in class applies here. Do not waste submissions to the Curator in testing your program! There is no point in submitting your program until you have verified that it produces correct results on the sample data files that are provided. If you waste all of your submissions because you have not tested your program adequately then you will receive a low score on this assignment. You will not be given extra submissions.

Your submitted program will be assigned a score, based upon the runtime testing performed by the Curator System. We will also be evaluating your submission of this program for documentation style and a few good coding practices. This will result in a deduction (ideally zero) that will be applied to your score from the Curator to yield your final score for this project.

Read the *Programming Standards* page on the CS 1044 website for general guidelines. You should comment your code in the same manner as the code given for the first two programming assignments. In particular:

- You should have a header comment identifying yourself, and describing what the program does.
- Every constant and variable you declare should have a comment explaining its logical significance in the program.
- Every function should have a header comment, as described in the notes and the *Programming Standards* page.
- Every major block of code should have a comment describing its purpose.
- Adopt a consistent indentation style and stick to it.

Your implementation must also meet the following requirements:

- Choose descriptive identifiers when you declare a variable or constant. Avoid choosing identifiers that are entirely lower-case.
- Use named constants instead of literal constants when the constant has logical significance.
- Use C++ streams for input and output, not C-style constructs.
- Use C++ string variables to hold character data, not C-style character pointers or arrays.
- Use C++ functions intelligently and appropriately. In particular, you must implement and use at least 4 functions in addition to `main()`. At least one of these functions must be `void`; at least one must return a value via

CS 1044 Program 5**Fall 2002**

return statement, and at least one must use a reference parameter to return a value. It is not acceptable that all function calls are from `main()` to other functions.

- Use the appropriate protocol when passing each parameter to a function. In particular, do not pass any parameter by reference unless it is being used to communicate a changed value from the called function to its caller.
- In a good design, each function is responsible for performing a single, constrained task, or for managing other functions. To promote that, none of the functions in your implementation should contain more than 30 executable statements. Comments and variable and type declarations are not executable; everything else is.

Understand that the list of requirements here is not a complete repetition of the *Programming Standards* page on the course website. It is possible that requirements listed there will be applied, even if they are not listed here.

This project requires implement quite a few features. None of them are terribly difficult to achieve, but there are quite a few. As a general rule, if you do not manage to complete everything, it is better to handle some of the commands completely correctly and to ignore the rest. Of course, that won't produce a good score unless you're handling most of the commands correctly, but it will still be better than having a lot of "almost there" handlers but none or few that are really correct.

Finally... this project serves in some sense as an exit exam for CS 1044. Students who cannot complete this project have very little chance of passing CS 1704. In order to give you the best possible chance of succeeding, I have included a relatively detailed discussion of design and implementation. I have also made the due date for the project as late as possible and still give the TAs time to evaluate your submissions fairly. As a result, there will be a very small window for late submissions. No submissions of this project will be accepted after midnight on the last day of classes.

Submitting your program:

You will submit this assignment to the Curator System (read the *Student Guide*), and it will be graded automatically. Instructions for submitting, and a description of how the grading is done, are contained in the *Student Guide*.

You will be allowed up to five submissions for this assignment. Use them wisely. Test your program thoroughly before submitting it. Make sure that your program produces correct results for every sample input file posted on the course website. If you do not get a perfect score, analyze the problem carefully and test your fix with the input file returned as part of the Curator e-mail message, before submitting again. The highest score you achieve will be counted.

The *Student Guide* and submission link can be found at: <http://www.cs.vt.edu/curator/>

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
// On my honor:
//
// - I have not discussed the C++ language code in my program with
//   anyone other than my instructor or the teaching assistants
//   assigned to this course.
//
// - I have not used C++ language code obtained from another student,
//   or any other unauthorized source, either modified or unmodified.
//
// - If any C++ language code or documentation used in my program
//   was obtained from another source, such as a text book or course
//   notes, that has been clearly noted with a proper citation in
//   the comments of my program.
//
// - I have not designed this program in such a way as to defeat or
//   interfere with the normal operation of the Curator System.
//
```

CS 1044 Program 5

Fall 2002

// <Student Name>

Failure to include this pledge in a submission is a violation of the Honor Code.**Sample report:**

The log file must be named QueryLog.txt. Here is a sample log file that corresponds to the input data given above:

```

Programmer: Bill McQuain
Portfolio Database

-----
show Microsft
  0:           Microsft    467    45.67    44.62    20837.54
-----
net change Microsft
-490.35
-----
sell 100 of Microsft
Sold for 4462.00
-----
show Microsft
  0:           Microsft    367    45.67    44.62    16375.54
-----
total value
128372.54
-----
Buy 200 of Microsft
Bought for      8924.00
-----
show Microsft
  0:           Microsft    567    45.67    44.62    25299.54
-----
show Citi
Not found
-----
quote DellCptr    22.50
DellCptr updated to 22.50
-----
show DellCptr
  3:           DellCptr    811    24.32    22.50    18247.50
-----
sort by name
Swaps: 5
-----
display
  0:           AT&T    1017    9.31    6.12    6224.04
  1:           CSX    404    33.66    41.32    16693.28
  2:           DellCptr    811    24.32    22.50    18247.50
  3:           IBM    985    67.90    71.01    69944.85
  4:           Lucent    413    1.60    4.23    1746.99
  5:           Microsft    567    45.67    44.62    25299.54
-----
sort by value
Swaps: 5
-----
display
  0:           Lucent    413    1.60    4.23    1746.99

```

CS 1044 Program 5

Fall 2002

```
1:          AT&T    1017    9.31    6.12    6224.04
2:          CSX     404    33.66   41.32   16693.28
3:      DellCptr   811    24.32   22.50   18247.50
4:      Microsft   567    45.67   44.62   25299.54
5:          IBM     985    67.90   71.01   69944.85
```

```
-----
exit
Exiting script processing.
-----
```

This page intentionally left blank.

Appendix F

Post-Project Survey(s)

Post-project User Surveys

About You...

What is your VT PID? (Please don't enter your SSN and DO NOT LEAVE BLANK!)

The Programming Environment

Did you use any other programming environment (other than .NET) at all in completing the project? (Yes/No)

How would you categorize your experience in starting a new project/program in the programming environment you used? (Very Easy, Easy, Medium, Difficult, Very Difficult)

How would you categorize the difficulty in entering code and debugging syntax or semantic bugs in your project solution using this programming environment? (Very Easy, Easy, Medium, Difficult, Very Difficult)

How would you categorize the difficulty in testing your project solution using this programming environment? (Very Easy, Easy, Medium, Difficult, Very Difficult)

Difficulty / Time and Work Required

How would you categorize the difficulty of the project? (Very Easy, Easy, Medium, Difficult, Very Difficult)

How many days did you work on the project? (Note: only count a day once when you worked on the project one or more times)

How many attempts at compilation did you perform while completing the project? (please quantify, don't enter "some")

How many attempts at execution did you perform in completing the project? (please quantify, don't enter "some")

Obtaining Help

(continued on next page)

Table F.1: *continued*

How often did you use the in-environment help (i.e. MSVC help, Sandbox help) over the course of completing the project? (Not at all, 1-5 times, (overall), 5-10 times (overall), 10-20 times (overall), All the time (so many, I can't think to count!))

Which of the following did you utilize for help in completing the project? (check all that apply) (A friend, A teaching assistant (grad or undergrad; DePasquale is a grad TA), The forum.cs.vt.edu web site, The instructor, The course note pack, The course text book (Programming and Problem Solving in C++), Other(s) (please specify):)

Other comments - (optional)

What features of the environment did you like the most, or were most helpful to you?

What features of the environment did you dislike the most, or caused you the most confusion?

Appendix G

Error Recognition Experiment Handout

CS1 .NET/Sandbox Error Recognition Experiment

Name _____ Environment: .NET SW SWO

Instructions: For each of the programs or code fragments below, identify all errors present or indicate that none exist. Assume that all code is as-is for this experiment, thus, point out ANYTHING you think is wrong (syntax, semantic, logic, or run time error).

```
#include <iostream>
using namespace std

int main () {
    cout << "My name is Peter DePasquale";
    cout << "I am a student at Virginia Tech!";
    return 0;
}
```

Errors (if any):

```
#include <iostream>
using namespace std;

int main() {
    int intValue = 100;

    while (intValue > -5) {
        cout << "The value of intValue is: " << intValue << endl;
        intValue = intValue - 5;
    }
    return 0;
}
```

Errors (if any):

```
#include <fstream>
#include <string>
using namespace std;

int main() {
    string myName = "Peter DePasquale";
    string myOffice = "McByrde 521";

    ofstream outFile;
    outFile.open("program.output");
    outFile << myName << '\t' << "(" << myOffice << ')' << endl;
    outFile.close();
    return 0;
}
```

Errors (if any):

```
#include <iostream>
using namespace std;

int main () {
    int cv = calcValue (5, 4);
    cout << "Calc value is " << cv << endl;
    return 0;
}

int calcValue (int first, int second) {
    return first * (second + first);
}
```

Errors (if any):

```
#include <iostream>
#include <string>
using namespace std;

int main () {
    int aValue = 1, bValue = 2, cValue = 3;
    string aString = "A string", bString = "B string";
    bool boolValue = true;

    cout << aString << " " << bValue << endl;
    cout << bString << " " << cValue << endl;
    cout << aValue << dValue << endl;
    cout << boolValue << endl;
    return 0;
}
```

Errors (if any):

```
#include <iostream>
using namespace std;

int main () {
    int loopVar = 100;
    boolean errorPresent = false;

    while (loopVar < 105 && !errorPresent) {
        cout << "loopVar is " << loopVar << endl;
        if ((loopVar % 5) == 0)
            loopVar = loopVar - 5;
        else
            errorPresent = true;
    }
    return 0;
}
```

Errors (if any):

```
#include <iostream>
using namespace std;

int main () {
    int xValue = 10, jValue = 15;

    for (int loopControl=0; loopControl < jValue; xValue++)
        cout << "xValue = " << xValue << " " << loopControl++ << endl;
        cout << "jValue = " << jValue << endl;
    return 0;
}
```

Errors (if any):

```
#include <iostream>
using namespace std;

int main () {
    int numberOfStudents = 100;
    int sumOfGrades = 8342;
    int numberOfTasks = 0;
    int numberOfProjects = 3;
    double numberOfTests = 3;
    double numberOfQuizzes = 2;

    cout << "Number of tasks: "
         << numberOfProjects + numberOfQuizzes + numberOfTests
         << endl;

    cout << "Number of students: " + numberOfStudents << endl;
    cout << "Sum of grades: " + sumOfGrades << endl;
    cout << "Average grade: " + sumOfGrades / numberOfTasks << endl;

    return 0;
}
```

Errors (if any):

```
#include <fstream>
#include <string>
using namespace std;

void outputData (string, int, double, ofstream);

int main () {
    ofstream outf;
    outf.open("output.data");
    outputData ("Peter DePasquale", 6964, 85.67, outf);
    outf.close();
    return 0;
}

void outputData (string name, int number, double grade, ofstream outFile) {
    outFile << "Student name: " << name << endl;
    outFile << "Student number: " << number << endl;
    outFile << "Student grade: " << grade << endl;
}
```

Errors (if any):

```
#include <iostream>
using namespace std;

int returnMult (int first, int second) {
    int int;

    int = first * second;

    return int;
}

int main () {
    cout << returnMult(5, 10) << endl;
    return 0;
}
```

Errors (if any):

```
#include <iostream>
using namespace std;

int main ( {
    cout << "Welcome to CS 1044";
    cout << "at Virginia Tech!";
    cout << "Your teaching assistants will be: Peter and Ines." << endl;
    return 0;
}
```

Errors (if any):

```
#include <iostream>
using namespace std;

int main () {
    ofstream outFile;
    outFile.open("classData.text");
    outFile << "Peter DePasquale " << "senior" << endl;
    outFile << "Missy Grant " << "junior" << endl;
    outFile << "Carolyn Wine " << "senior" << endl;
    outFile.close();
    return 0;
}
```

Errors (if any):

```
#include <iostream>
using namespace std;

int main () {
int index = 0;
for (index=0; index < 20; index++) {
cout << index * 5 << ": ";
cout << index - 5 << endl;
}
cout << endl;
for (index=100; index > 20; index=index-10) {
cout << index * 5 << ": ";
cout << index - 5 << endl;
}
cout << endl;
for (index=0; index < 20; index++) {
cout << index * 5 << ": ";
cout << index - 5 << endl;
}
cout << endl;
return 0;
}
```

Errors (if any):

Appendix H

Subset Configuration File

```
<?xml version="1.0"?>
<!DOCTYPE RestrictionInfo
[
  <!ELEMENT RestrictionInfo (Description?,Level*)>

  <!-- Level must have 0+ Categories -->
  <!ELEMENT Level (Description?, Category*)>

  <!-- Category must have at least one LanguageItem -->
  <!ELEMENT Category (Description?,LanguageItem+)>

  <!-- LanguageItems can have a description -->
  <!ELEMENT LanguageItem (Description?)>

  <!-- Descriptions describe the each LanguageItem -->
  <!ELEMENT Description (#PCDATA)>

  <!-- Higher levels must have names -->
  <!ATTLIST Level name CDATA #REQUIRED>
  <!ATTLIST Category name CDATA #REQUIRED>
  <!ATTLIST LanguageItem name CDATA #REQUIRED>
]
>
<RestrictionInfo>
  <Level name="Labs 1/2">
    <Category name="MULTIPLICATIVE">
      <LanguageItem name="multiplication" />
      <LanguageItem name="division" />
      <LanguageItem name="modulus" />
    </Category>

    <Category name="ADDITIVE">
      <LanguageItem name="addition" />
      <LanguageItem name="subtraction" />
    </Category>

    <Category name="SHIFT">
      <LanguageItem name="left" />
      <LanguageItem name="right" />
    </Category>

    <Category name="RELATIONAL">
      <LanguageItem name="less than" />
      <LanguageItem name="less than equal to" />
      <LanguageItem name="greater than" />
      <LanguageItem name="greater than equal to" />
    </Category>

    <Category name="EQUALITY">
      <LanguageItem name="=" (equality)" />
      <LanguageItem name="!=" (inequality)" />
    </Category>
  </Level>
</RestrictionInfo>
```

```

</Category>

<Category name="ASSIGNMENT">
  <LanguageItem name="=" />
</Category>

<Category name="DECLARATIONS">
  <LanguageItem name="boolean" />
  <LanguageItem name="char" />
  <LanguageItem name="short" />
  <LanguageItem name="int" />
  <LanguageItem name="long" />
  <LanguageItem name="double" />
</Category>

<Category name="JUMP">
  <LanguageItem name="return" />
</Category>

<Category name="LIBRARIES">
  <LanguageItem name="climits" />
  <LanguageItem name="cmath" />
  <LanguageItem name="fstream" />
  <LanguageItem name="iostream" />
</Category>
</Level>

<Level name="Project 1/Labs 3 and 4">
  <Category name="EXPRESSIONS">
    <LanguageItem name="typecast" />
  </Category>

  <Category name="MULTIPLICATIVE">
    <LanguageItem name="multiplication" />
    <LanguageItem name="division" />
    <LanguageItem name="modulus" />
  </Category>

  <Category name="ADDITIVE">
    <LanguageItem name="addition" />
    <LanguageItem name="subtraction" />
  </Category>

  <Category name="SHIFT">
    <LanguageItem name="left" />
    <LanguageItem name="right" />
  </Category>

  <Category name="RELATIONAL">
    <LanguageItem name="less than" />
    <LanguageItem name="less than equal to" />
    <LanguageItem name="greater than" />
    <LanguageItem name="greater than equal to" />
  </Category>

  <Category name="EQUALITY">
    <LanguageItem name=="= (equality)" />
    <LanguageItem name=="!= (inequality)" />
  </Category>

  <Category name="ASSIGNMENT">
    <LanguageItem name="=" />
    <LanguageItem name="*" />
    <LanguageItem name="/" />
    <LanguageItem name="%" />
    <LanguageItem name="+=" />
    <LanguageItem name="--" />
  </Category>

  <Category name="DECLARATIONS">
    <LanguageItem name="boolean" />
    <LanguageItem name="char" />
    <LanguageItem name="short" />
    <LanguageItem name="int" />
  </Category>

```

```

    <LanguageItem name="long" />
    <LanguageItem name="double" />
    <LanguageItem name="const" />
</Category>

<Category name="SELECTION">
    <LanguageItem name="if" />
    <LanguageItem name="if-else" />
    <LanguageItem name="switch" />
</Category>

<Category name="JUMP">
    <LanguageItem name="break" />
    <LanguageItem name="return" />
</Category>

<Category name="LIBRARIES">
    <LanguageItem name="climits" />
    <LanguageItem name="cmath" />
    <LanguageItem name="cstdlib" />
    <LanguageItem name="fstream" />
    <LanguageItem name="iomanip" />
    <LanguageItem name="iostream" />
    <LanguageItem name="string" />
</Category>
</Level>

<Level name="Project 2/Labs 5-7">
    <Category name="EXPRESSIONS">
        <LanguageItem name="typecast" />
    </Category>

    <Category name="MULTIPLICATIVE">
        <LanguageItem name="multiplication" />
        <LanguageItem name="division" />
        <LanguageItem name="modulus" />
    </Category>

    <Category name="ADDITIVE">
        <LanguageItem name="addition" />
        <LanguageItem name="subtraction" />
    </Category>

    <Category name="SHIFT">
        <LanguageItem name="left" />
        <LanguageItem name="right" />
    </Category>

    <Category name="RELATIONAL">
        <LanguageItem name="less than" />
        <LanguageItem name="less than equal to" />
        <LanguageItem name="greater than" />
        <LanguageItem name="greater than equal to" />
    </Category>

    <Category name="EQUALITY">
        <LanguageItem name=="=" (equality)" />
        <LanguageItem name!="=" (inequality)" />
    </Category>

    <Category name="LOGICAL">
        <LanguageItem name="&& (and)" />
        <LanguageItem name="|| (or)" />
    </Category>

    <Category name="ASSIGNMENT">
        <LanguageItem name="=" />
        <LanguageItem name="=" />
        <LanguageItem name="/=" />
        <LanguageItem name="%=" />
        <LanguageItem name="+=" />
        <LanguageItem name="-=" />
    </Category>

```

```

<Category name="DECLARATIONS">
  <LanguageItem name="boolean" />
  <LanguageItem name="char" />
  <LanguageItem name="short" />
  <LanguageItem name="int" />
  <LanguageItem name="long" />
  <LanguageItem name="double" />
  <LanguageItem name="const" />
</Category>

<Category name="SELECTION">
  <LanguageItem name="if" />
  <LanguageItem name="if-else" />
  <LanguageItem name="switch" />
</Category>

<Category name="ITERATION">
  <LanguageItem name="do-while" />
  <LanguageItem name="while" />
  <LanguageItem name="for" />
</Category>

<Category name="JUMP">
  <LanguageItem name="break" />
  <LanguageItem name="return" />
</Category>

<Category name="LIBRARIES">
  <LanguageItem name="climits" />
  <LanguageItem name="cmath" />
  <LanguageItem name="cstdlib" />
  <LanguageItem name="fstream" />
  <LanguageItem name="iomanip" />
  <LanguageItem name="iostream" />
  <LanguageItem name="string" />
</Category>
</Level>

<Level name="Project 3/Labs 8-9">
  <Category name="EXPRESSIONS">
    <LanguageItem name="prefix" />
    <LanguageItem name="postfix" />
    <LanguageItem name="typecast" />
  </Category>

  <Category name="MULTIPLICATIVE">
    <LanguageItem name="multiplication" />
    <LanguageItem name="division" />
    <LanguageItem name="modulus" />
  </Category>

  <Category name="ADDITIVE">
    <LanguageItem name="addition" />
    <LanguageItem name="subtraction" />
  </Category>

  <Category name="SHIFT">
    <LanguageItem name="left" />
    <LanguageItem name="right" />
  </Category>

  <Category name="RELATIONAL">
    <LanguageItem name="less than" />
    <LanguageItem name="less than equal to" />
    <LanguageItem name="greater than" />
    <LanguageItem name="greater than equal to" />
  </Category>

  <Category name="EQUALITY">
    <LanguageItem name="==" (equality) />
    <LanguageItem name!=" (inequality) />
  </Category>

  <Category name="LOGICAL">

```

```
<LanguageItem name="&& (and)" />
<LanguageItem name="|| (or)" />
</Category>

<Category name="ASSIGNMENT">
  <LanguageItem name="=" />
  <LanguageItem name="=" />
  <LanguageItem name="/=" />
  <LanguageItem name="%=" />
  <LanguageItem name="+=" />
  <LanguageItem name="-=" />
</Category>

<Category name="DECLARATIONS">
  <LanguageItem name="boolean" />
  <LanguageItem name="char" />
  <LanguageItem name="short" />
  <LanguageItem name="int" />
  <LanguageItem name="long" />
  <LanguageItem name="double" />
  <LanguageItem name="const" />
</Category>

<Category name="SELECTION">
  <LanguageItem name="if" />
  <LanguageItem name="if-else" />
  <LanguageItem name="switch" />
</Category>

<Category name="ITERATION">
  <LanguageItem name="do-while" />
  <LanguageItem name="while" />
  <LanguageItem name="for" />
  <LanguageItem name="for with declarations" />
</Category>

<Category name="JUMP">
  <LanguageItem name="break" />
  <LanguageItem name="return" />
</Category>

<Category name="LIBRARIES">
  <LanguageItem name="climits" />
  <LanguageItem name="cmath" />
  <LanguageItem name="cstdlib" />
  <LanguageItem name="fstream" />
  <LanguageItem name="iomanip" />
  <LanguageItem name="iostream" />
  <LanguageItem name="string" />
</Category>
</Level>
</RestrictionInfo>
```

This page intentionally left blank.

Appendix I

Grammar Recognized by the *CS1 Sandbox*

Introduction

This syntax summary originally was part of the ISO/IEC C99 Language Standard [82]. Various portions of the language were removed for the prototype system to support the CS1044 sections which were used as an experimentation base for this dissertation.

Section names prefixed with "C++" are from the C++ Language standard, ISO/IEC 14882 (1998-09-01). The Appendix entry that they're from is listed as part of the section name. For example, "(C++.A.6)" is listed in Appendix A.6 of the C++ standard.

Lexical elements

(6.4) *token*:

keyword
identifier
constant
string-literal
punctuator
boolean-literal

(C++.A.2) *boolean-literal*:

false
true

(6.4) *preprocessing-token*:

header-name
identifier
pp-number
character-constant
string-literal
punctuator

Keywords

(6.4.1) *keyword*: one of

break	else	short
case	enum	sizeof
char	float	struct
const	for	switch
continue	if	typedef
default	int	void
do	long	while
double	return	
bool	namespace	using

Identifiers

(6.4.2.1) *identifier*:

identifier-nondigit
identifier identifier-nondigit
identifier digit

(6.4.2.1) *identifier-nondigit*:

nondigit
 other implementation-defined characters

(6.4.2.1) *nondigit*: one of

_ a b c d e f g h i j k l m
n o p q r s t u v w x y z
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

(6.4.2.1) *digit*: one of

0 1 2 3 4 5 6 7 8 9

Constants

(6.4.4) *constant*:

integer-constant
floating-constant
enumeration-constant
character-constant

(6.4.4.1) *integer-constant*:

decimal-constant

(6.4.4.1) *decimal-constant*:

nonzero-digit
decimal-constant digit

(6.4.4.1) *nonzero-digit*: one of

1 2 3 4 5 6 7 8 9

(6.4.4.2) *floating-constant*:

decimal-floating-constant

(6.4.4.2) *decimal-floating-constant*:

fractional-constant exponent-part_{opt} floating-suffix_{opt}
digit-sequence exponent-part floating-suffix_{opt}

(6.4.4.2) *fractional-constant*:

digit-sequence_{opt} . digit-sequence
digit-sequence .

(6.4.4.2) *exponent-part*:

e *sign_{opt} digit-sequence*
E *sign_{opt} digit-sequence*

(6.4.4.2) *sign*: one of

+ -

(6.4.4.2) *digit-sequence*:

digit
digit-sequence digit

(6.4.4.2) *floating-suffix*: one of

f F

(6.4.4.3) *enumeration-constant*:

identifier

(6.4.4.4) *character-constant*:

'c-char-sequence'

(6.4.4.4) *c-char-sequence*:

c-char
c-char-sequence c-char

(6.4.4.4) *c-char*:

any member of the source character set except the single-quote
 ', backslash \, or new-line character
escape-sequence

(6.4.4.4) *escape-sequence*:

simple-escape-sequence

(6.4.4.4) *simple-escape-sequence*: one of

`\' \"` `\?`
`\a \b \f`
`\n \r \t \v`

String literals

(6.4.5) *string-literal*:

`" s-char-sequence_opt "`

(6.4.5) *s-char-sequence*:

s-char
s-char-sequence s-char

(6.4.5) *s-char*:

any member of the source character set except the double-quote `"`, backslash `\`, or new-line character *escape-sequence*

Punctuators

(6.4.6) *punctuator*: one of

<code>[</code>	<code>]</code>	<code>(</code>	<code>)</code>	<code>{</code>	<code>}</code>	<code>.</code>	<code>!</code>
<code>++</code>	<code>--</code>	<code>&</code>	<code>*</code>	<code>+</code>	<code>-</code>	<code>~</code>	
<code>/</code>	<code>%</code>	<code><<</code>	<code>>></code>	<code><</code>	<code>></code>	<code><=</code>	
<code>>=</code>	<code>==</code>	<code>!=</code>	<code>^</code>	<code> </code>	<code>&&</code>	<code> </code>	
<code>?</code>	<code>:</code>	<code>;</code>	<code>...</code>	<code>&=</code>	<code>^=</code>	<code> =</code>	
<code>=</code>	<code>*=</code>	<code>/=</code>	<code>%=</code>	<code>+=</code>	<code>-=</code>	<code><<=</code>	<code>>>=</code>
<code>,</code>	<code>#</code>	<code>##</code>					

Header names

(6.4.7) *header-name*:

`< h-char-sequence >`
`" q-char-sequence "`

(6.4.7) *h-char-sequence*:

h-char
h-char-sequence h-char

(6.4.7) *h-char*:

any member of the source character set except the new-line character and `>`

(6.4.7) *q-char-sequence*:

q-char
q-char-sequence q-char

(6.4.7) *q-char*:

any member of the source character set except the new-line character and ”

Expressions

(6.5.1) *primary-expr*:

identifier
constant
string-literal
 (*expression*)

(6.5.2) *postfix-expr*:

primary-expr postfix-expr [*expression*]
type-specifier (*expression*)
postfix-expr (*argument-expr-list*_{opt})
postfix-expr . *identifier*
postfix-expr ++
postfix-expr –
 (*type-name*) { *initializer-list* }
 (*type-name*) { *initializer-list* , }
reinterpret_cast < *type-id* > (*expression*)
const_cast < *type-id* > (*expression*)

(C++A.7) *type-id*:

type-name

(6.5.2) *argument-expr-list*:

assignment-expr
argument-expr-list , *assignment-expr*

(6.5.3) *unary-expr*:

postfix-expr
 ++ *unary-expr*
 – *unary-expr*
unary-operator cast-expr
sizeof *unary-expr*
sizeof (*type-name*)

(6.5.4) *cast-expr*:

unary-expr
 (*type-name*) *cast-expr*

(6.5.5) *multiplicative-expr*:

cast-expr
multiplicative-expr * *cast-expr*
multiplicative-expr / *cast-expr*
multiplicative-expr % *cast-expr*

(6.5.6) *additive-expr*:

multiplicative-expr
additive-expr + *multiplicative-expr*
additive-expr - *multiplicative-expr*

(6.5.7) *shift-expr*:

additive-expr
shift-expr << *additive-expr*
shift-expr >> *additive-expr*

(6.5.8) *relational-expr*:

shift-expr
relational-expr < *shift-expr*
relational-expr > *shift-expr*
relational-expr <= *shift-expr*
relational-expr >= *shift-expr*

(6.5.9) *equality-expr*:

relational-expr
equality-expr == *relational-expr*
equality-expr != *relational-expr*

(6.5.13) *logical-AND-expr*:

inclusive-OR-expr
logical-AND-expr && *inclusive-OR-expr*

(6.5.14) *logical-OR-expr*:

logical-AND-expr
logical-OR-expr —— *logical-AND-expr*

(6.5.16) *assignment-expr*:

conditional-expr
unary-expr *assignment-operator* *assignment-expr*

(6.5.16) *assignment-operator*: one of

= *= /= %= += -=

(6.5.17) *expression*:

assignment-expr
expression , *assignment-expr*

(6.6) *constant-expr*:

conditional-expr

Declarations

(6.7) *declaration*:

block-declaration ;
declaration-specifiers *init-declarator-list*_{opt} ;

(C++ .A.6) *block-declaration*:

using-declaration
using-directive

(C++ .A.6) *using-declaration*:

using ::_{opt} *nested-name-specifier*_{opt} *namespace-name*

(C++ .A.6) *using-directive*:

using namespace ::_{opt} *nested-name-specifier*_{opt} *namespace-name*

(C++ .A.6) *nested-name-specifier*:

class-or-namespace-name :: *nested-name-specifier*_{opt}

(C++ .A.6) *class-or-namespace-name*

class-name
namespace-name

(C++ .A.8) *class-name*

identifier

(C++ .A.6) *namespace-name*

original-namespace-name
namespace-alias

(C++ .A.6) *original-namespace-name*

identifier

(C++ .A.6) *namespace-alias*

identifier

(6.7) *declaration-specifiers*:

storage-class-specifier *declaration-specifiers*_{opt}
type-specifier *declaration-specifiers*_{opt}
type-qualifier *declaration-specifiers*_{opt}
function-specifier *declaration-specifiers*_{opt}
ptr-operator *declaration-specifier*

(C++ .A.7) *ptr-operator*:

&

(6.7) *init-declarator-list*:

init-declarator
init-declarator-list , *init-declarator*

(6.7) *init-declarator*:

declarator
declarator = *initializer*

(6.7.1) *storage-class-specifier*:

typedef

(6.7.2) *type-specifier*:

void
bool
char
short
int
long
float
double
struct-or-union-specifier
enum-specifier
typedef-name
bool

(6.7.2.1) *struct-or-union-specifier*:

*struct-or-union identifier*_{opt} { *struct-declaration-list* }
struct-or-union identifier

(6.7.2.1) *struct-or-union*:

struct

(6.7.2.1) *struct-declaration-list*:

struct-declaration
struct-declaration-list *struct-declaration*

(6.7.2.1) *struct-declaration*:

specifier-qualifier-list *struct-declarator-list* ;

(6.7.2.1) *specifier-qualifier-list*:

type-specifier *specifier-qualifier-list*_{opt}
type-qualifier *specifier-qualifier-list*_{opt}

(6.7.2.1) *struct-declarator-list*:

struct-declarator
struct-declarator-list , *struct-declarator*

(6.7.2.1) *struct-declarator*:

declarator

(6.7.2.2) *enum-specifier*:

enum *identifier*_{opt} } *enumerator-list* }
enum *identifier*_{opt} } *enumerator-list* , }
enum *identifier*

(6.7.2.2) *enumerator-list*:

enumerator
enumerator-list , *enumerator*

(6.7.2.2) *enumerator*:

enumeration-constant
enumeration-constant = *constant-expr*

(6.7.3) *type-qualifier*:

const

(6.7.5) *declarator*:

direct-declarator

(6.7.5) *direct-declarator*:

identifier
(*declarator*)
direct-declarator (*parameter-type-list*)
direct-declarator (*identifier-list*_{opt})

(6.7.5) *type-qualifier-list*:

type-qualifier
type-qualifier-list *type-qualifier*

(6.7.5) *parameter-type-list*:

parameter-list

(6.7.5) *parameter-list*:

parameter-declaration *parameter-list* , *parameter-declaration*

(6.7.5) *parameter-declaration*:

declaration-specifiers *declarator*

(6.7.5) *identifier-list*:

identifier
identifier-list , *identifier*

(6.7.6) *type-name*:

specifier-qualifier-list

(6.7.7) *typedef-name*:

identifier

(6.7.8) *initializer*:

assignment-expr
 { *initializer-list* }
 { *initializer-list* , }

(6.7.8) *initializer-list*:

designation_{opt} initializer
initializer-list , *designation_{opt} initializer*

(6.7.8) *designation*:

designator-list =

(6.7.8) *designator-list*:

designator designator-list designator

(6.7.8) *designator*:

[*constant-expr*]
 . *identifier*

Statements

(6.8) *statement*:

labeled-statement
compound-statement
expression-statement
selection-statement
iteration-statement
jump-statement

(6.8.1) *labeled-statement*:

case *constant-expr* : *statement*
default : *statement*

(6.8.2) *compound-statement*:

{ *block-item-list_{opt}* }

(6.8.2) *block-item-list*:

block-item
block-item-list block-item

(6.8.2) *block-item*:

declaration
statement

(6.8.3) *expression-statement*:

*expression*_{opt} ;

(6.8.4) *selection-statement*:

if (*expression*) *statement*
if (*expression*) *statement* **else** *statement*
switch (*expression*) *statement*

(6.8.5) *iteration-statement*:

while (*expression*) *statement*
do *statement* **while** (*expression*) ;
for (*expression*_{opt} ; *expression*_{opt} ;
*expression*_{opt}) *statement*
for (*declaration* *expression*_{opt} ; *expression*_{opt}) *statement*

(6.8.6) *jump-statement*:

continue ;
break ;
return *expression*_{opt} ;

External definitions

(6.9) *translation-unit*:

external-declaration
translation-unit *external-declaration*

(6.9) *external-declaration*:

function-definition
declaration

(6.9.1) *function-definition*:

declaration-specifiers *declarator* *declaration-list*_{opt} *compound-statement*

(6.9.1) *declaration-list*:

declaration
declaration-list *declaration*

Preprocessing directives

(6.10) *preprocessing-file*:

*group*_{opt}

(6.10) *group*:

group-part
group group-part

(6.10) *group-part*:

control-line

(6.10) *control-line*:

include *pp-tokens new-line*

(6.10) *new-line*:

the new-line character

Appendix J

ANTLR Input and Output Files

J.1 ANTLR Input File Used to Create *CS1 Sandbox*'s Lexer (CLexer.g)

```

header {
    package sandbox.compiler.frontend;

    import java.util.*;
    import sandbox.lib.*;
    import sandbox.compiler.*;
    import sandbox.client.ErrorCodeTable;
}

/**
 * The lexer is the front-end to the parser. It "tokenizes" the input,
 * splitting the original input into "tokens" that the grammar understands.
 */
class CLexer extends Lexer;

options {
    k = 3;
    charVocabulary = '\3'..'\'377';
    exportVocab = CLexer;
}

tokens {
    DOT;
    FLOATING_CONSTANT;

/* (6.4.1) keyword: one of

        break      enum      struct
        case       float     switch
        char       for       typedef
        const      if        void
        continue   int       while
        default    long
        do         return
        double     short
        else       sizeof

        bool       using     namespace
*/
// Primitive types (including "void") are added to the symbol
// table directly in TypeManager.java, and not handled here

BREAK = "break";
CASE = "case";
CONST = "const";
CONST_CAST = "const_cast";
CONTINUE = "continue";
DEFAULT = "default";
DO = "do";
ELSE = "else";
ENUM = "enum";
FOR = "for";
FALSE = "false";
IF = "if";
NAMESPACE = "namespace";
USING = "using";
RETURN = "return";
REINTERPRET_CAST = "reinterpret_cast";
SIZEOF = "sizeof";
STATIC_CAST = "static_cast";
STRUCT = "struct";
SWITCH = "switch";
TYPENAME;
TYPEDEF = "typedef";
TRUE = "true";
WHILE = "while";
}
{

```

```

/** The errors that are observed during the parse phase. */
private Vector errorVector = new Vector();

/** The manager that handles included libraries */
private LibraryManager libraryManager;

/** Sets the library manager to the parameter */
public void setLibraryManager(LibraryManager mgr) {
    libraryManager = mgr;
}

/**
 * Returns a vector of each error seen during compilation.
 */
public Vector getErrorVector() {
    return(errorVector);
}

public void reportError(RecognitionException ex){
    String errorMsg;

    AntlrError error = new AntlrError();
    error.setType("recognition exception - lexer error");

    // tack "-- lexer" onto the raw message so that it can be classified later
    errorMsg = new String(ex.getMessage() + "-- lexer");
    error.setErrorMessage(errorMsg);
    error.setLineNumber(ex.getLine());
    error.setColumnNumber(ex.getColumn());
    errorVector.add(error);
}

public void reportError(String s) {
    AntlrError error = new AntlrError();
    error.setType("error");
    error.setErrorMessage(s);
    errorVector.add(error);
}

public void reportWarning(String s) {
    AntlrError error = new AntlrError();
    error.setType("warning");
    error.setErrorMessage(s);
    errorVector.add(error);
}

public void reportError(String s, int line, int column, int errorCode) {
    AntlrError error = new AntlrError();
    error.setType("error");
    error.setErrorMessage(s);
    error.setLineNumber(line);
    error.setColumnNumber(column);
    error.setErrorCode(errorCode);
    errorVector.add(error);
}
}

/* Punctuation:
(6.4.6) punctuator: one of
    [ ] ( ) { } . ->
    ++ -- & * + - ~ !
    / % << >> < > <= >= == != ^ | && ||
    ? : ; ...
    = *= /= %= += -= <<= >>= &= ^= |=
    , # ##
*/

LBRACKET : '[' ;
RBRACKET : ']' ;

```

```

LPAREN : '(' ;
RPAREN : ')' ;
LBRACE : '{' ;
RBRACE : '}' ;
/* DOT is processed specially in INTEGER_LITERAL */
INCR : "++" ;
DECR : "--" ;
AMPERSAND : '&' ;
STAR : '*' ;
PLUS : '+' ;
MINUS : '-' ;
TILDE : '~' ;
BANG : '!' ;
SLASH : '/' ;
PERCENT : '%' ;
LSHIFT : "<<" ;
RSHIFT : ">>" ;
LT : '<' ;
GT : '>' ;
LE : "<=" ;
GE : ">=" ;
EQ : "==" ;
NE : "!=" ;
CARET : '^' ;
PIPE : '|' ;
LAND : "&&" ;
LOR : "||" ;
QUESTION : '?' ;
COLON : ':' ;
COLON_COLON : "::" ;
SEMI : ';' ;
ASSIGN : '=' ;
STAR_ASSIGN : "*=" ;
SLASH_ASSIGN : "/=" ;
PERCENT_ASSIGN : "%=" ;
PLUS_ASSIGN : "+=" ;
MINUS_ASSIGN : "-=" ;
COMMA : ',' ;
POUND : '#' ;

```

```

/** These currently seem to be unused. */
LSHIFT_ASSIGN : "<<=" ;
RSHIFT_ASSIGN : ">>=" ;
AMPERSAND_ASSIGN : "&=" ;
CARET_ASSIGN : "^=" ;
PIPE_ASSIGN : "|=" ;

```

```

/* Whitespace */
WS
: ( WHITESPACE
  | NEW_LINE
  )
  { _ttype = Token.SKIP; }
;

```

```

/* whitespace that doesn't include new-line. */
protected
WHITESPACE
: ' '
| '\ t'
| '\ f'
;

```

```

/* newlines supported */
protected
NEW_LINE
: "\ r\ n" { newline(); }
| '\ r' { newline(); }
| '\ n' { newline(); }
;

```

```

/* Single-line comments */

```

```

SL_COMMENT
: "//" (~('\ n'|'\ r'))* ('\ n'|'\ r'('\ n'))?
  {$setType(Token.SKIP); newline();}
;

/* Multi-line comments (such as this one) */
ML_COMMENT
: "/*"
  (
    options {
      generateAmbigWarnings=false;
    }
    :
    { LA(2)!='/' }? '*'
  | '\ r' '\ n'      {newline();}
  | '\ r'           {newline();}
  | '\ n'          {newline();}
  | ~('*'|'\ n'|'\ r')
  )*
  "*/"
  {$setType(Token.SKIP);}
;

/* A.1.3 Identifiers */
IDENTIFIER
options { testLiterals = false; }
: IDENTIFIER_NONDIGIT (IDENTIFIER_NONDIGIT | DIGIT)*
  { if(libraryManager.getTypeManager().isTypeName($getText)) $setType(TYPENAME); }
;

protected
IDENTIFIER_NONDIGIT : 'A'..'Z' | 'a'..'z' | '_' ;

protected
DIGIT : '0'..'9' ;

/* A.1.5 Constants */

/* This handles FLOATING_CONSTANT as well (see JavaRecognizer for example) */
INTEGER_CONSTANT
{ boolean isDouble = true, isLong = false; }
: '.' { _ttype = DOT; }
  (
    (DIGIT)+ (EXPONENT)? (FLOATING_SUFFIX { isDouble = false; } )?
  {
    CToken ct = new CToken(FLOATING_CONSTANT, $getText);

    try {
      if(isDouble)
        ct.setValue(Double.valueOf($getText));
      else
        ct.setValue(Float.valueOf($getText));
    } catch(NumberFormatException ex) { }
    $setToken(ct);
  }
  )?
| (DIGIT)+
  {
    CToken ict = new CToken(INTEGER_CONSTANT, $getText);

    try {
      ict.setValue(Integer.valueOf($getText));
    } catch(NumberFormatException ex) { }
    $setToken(ict);
  }
  (
    (
      INTEGER_SUFFIX!
    {
      try {
        ict.setValue(Long.valueOf($getText));
      } catch(NumberFormatException ex) { }
    }
  )
  )
  | (

```

```

    ( ('.' (DIGIT)+ (EXPONENT)?
      | EXPONENT
      ) (FLOATING_SUFFIX! { isDouble = false; } )?)?
  {
    CToken ct = new CToken(FLOATING_CONSTANT, $getText);

    try {
      if(isDouble)
        ct.setValue(Double.valueOf($getText));
      else
        ct.setValue(Float.valueOf($getText));
    } catch(NumberFormatException ex) { }
    $setToken(ct);
  }
)
)?
;

protected
EXPONENT
: ('e' | 'E') (PLUS | MINUS)? (DIGIT)+
;

protected
FLOATING_SUFFIX : 'f' | 'F' ;

protected
INTEGER_SUFFIX : 'l' | 'L' ;

CHAR_CONSTANT
: '\'' (ESC | ~'\'' ) '\''
{
  CToken ct = new CToken(CHAR_CONSTANT, $getText);

  ct.setValue(new Character(EscapeTranslator.translateEscapeSequence($getText).charAt(1)));
  $setToken(ct);
}
;

STRING_LITERAL
: '"' (ESC | ~('" | '\'))* '"'
{
  CToken ct = new CToken(STRING_LITERAL, $getText);

  ct.setValue(EscapeTranslator.translateEscapeSequence($getText));
  $setToken(ct);
}
;

/* Character escape codes */
protected
ESC
: '\\
( 'n'
| 'r'
| 't'
| 'b'
| 'f'
| '"'
| '\''
| '\\"
| '\\\'
| ('0'..'3')
(
  options {
    warnWhenFollowAmbig = false;
  }
: ('0'..'7')
(
  options {
    warnWhenFollowAmbig = false;
  }
: ('0'..'7')
)?
)?
)?

```

```

    | ('4'..'7')
      (
        options {
          warnWhenFollowAmbig = false;
        }
        : ('0'..'9')
      )?
    )
;

//
// Preprocessor Support.
//

protected
HEADER_NAME
: LT! (~('>' | '\ r' | '\ n'))* GT! // h-char-sequence
{
  CToken ct = new CToken(HEADER_NAME, $getText());

  ct.setValue($getText());
  $setToken(ct);
}
| '''! (~('"' | '\ r' | '\ n'))* '''! // q-char-sequence
{
  CToken ct = new CToken(HEADER_NAME, $getText());

  ct.setValue($getText());
  $setToken(ct);
}
;

protected
PP_TOKENS
: (PREPROCESSING_TOKEN)+
;

protected
PREPROCESSING_TOKEN
: HEADER_NAME
;

POUND_INCLUDE_DIRECTIVE
// Surrounding "include"! there needs to be a try/catch block which
// is hit when a pound (#) symbol is present but "include" does not follow it
: POUND! (WHITESPACE)*! "include"! (WHITESPACE)*! p:PP_TOKENS (WHITESPACE)*!
(NEW_LINE! | SL_COMMENT! | ML_COMMENT! NEW_LINE!)
{
  try {
    libraryManager.loadLibrary(p.getText());
  }
  catch(NoSuchLibraryException ex) {
    reportError("There is no library named " + p.getText() + "",
      getLine() - 1, getColumn(), ErrorCodeTable.INVALID_INCLUDE_FILE);
  }
  catch(LibraryInitializationException ex) {
    reportError("Error initializing library " + p.getText() + "",
      getLine() - 1, getColumn(), ErrorCodeTable.ERROR_INITIALIZING_INCLUDE);
  }
}
;

```

J.2 ANTLR-produced Output File for the *CS1 Sandbox*'s Lexer (CLexer.java)

```
// $ANTLR 2.7.1: "CLexer.g" -> "CLexer.java"$

package sandbox.compiler.frontend;

import java.util.*;
import sandbox.lib.*;
import sandbox.compiler.*;
import sandbox.client.ErrorCodeTable;

import java.io.InputStream;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.TokenStreamRecognitionException;
import antlr.CharStreamException;
import antlr.CharStreamIOException;
import antlr.ANTLRException;
import java.io.Reader;
import java.util.Hashtable;
import antlr.CharScanner;
import antlr.InputBuffer;
import antlr.ByteBuffer;
import antlr.CharBuffer;
import antlr.Token;
import antlr.CommonToken;
import antlr.RecognitionException;
import antlr.NoViableAltForCharException;
import antlr.MismatchedCharException;
import antlr.TokenStream;
import antlr.ANTLRHashString;
import antlr.LexerSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.SemanticException;

/**
 * The lexer is the front-end to the parser. It "tokenizes" the input,
 * splitting the original input into "tokens" that the grammar understands.
 */
public class CLexer extends antlr.CharScanner implements CLexerTokenTypes, TokenStream
{

    /** The errors that are observed during the parse phase. */
    private Vector errorVector = new Vector();

    /** The manager that handles included libraries */
    private LibraryManager libraryManager;

    /** Sets the library manager to the parameter */
    public void setLibraryManager(LibraryManager mgr) {
        libraryManager = mgr;
    }

    /**
     * Returns a vector of each error seen during compilation.
     */
    public Vector getErrorVector() {
        return(errorVector);
    }

    public void reportError(RecognitionException ex){
        String errorMsg;

        AntlrError error = new AntlrError();
        error.setType("recognition exception - lexer error");

        // tack " -- lexer" onto the raw message so that it can be classified later
        errorMsg = new String(ex.getMessage() + " -- lexer");
    }
}

```

```

        error.setErrorMessage(errorMsg);
        error.setLineNumber(ex.getLine());
        error.setColumnNumber(ex.getColumn());
        errorVector.add(error);
    }

    public void reportError(String s) {
        AntlrError error = new AntlrError();
        error.setType("error");
        error.setErrorMessage(s);
        errorVector.add(error);
    }

    public void reportWarning(String s) {
        AntlrError error = new AntlrError();
        error.setType("warning");
        error.setErrorMessage(s);
        errorVector.add(error);
    }

    public void reportError(String s, int line, int column, int errorCode) {
        AntlrError error = new AntlrError();
        error.setType("error");
        error.setErrorMessage(s);
        error.setLineNumber(line);
        error.setColumnNumber(column);
        error.setErrorCode(errorCode);
        errorVector.add(error);
    }
}

public CLexer(InputStream in) {
    this(new ByteBuffer(in));
}

public CLexer(Reader in) {
    this(new CharBuffer(in));
}

public CLexer(InputBuffer ib) {
    this(new LexerSharedInputState(ib));
}

public CLexer(LexerSharedInputState state) {
    super(state);
    literals = new Hashtable();
    literals.put(new ANTLRHashString("switch", this), new Integer(25));
    literals.put(new ANTLRHashString("case", this), new Integer(7));
    literals.put(new ANTLRHashString("reinterpret_cast", this), new Integer(21));
    literals.put(new ANTLRHashString("for", this), new Integer(15));
    literals.put(new ANTLRHashString("sizeof", this), new Integer(22));
    literals.put(new ANTLRHashString("false", this), new Integer(16));
    literals.put(new ANTLRHashString("true", this), new Integer(28));
    literals.put(new ANTLRHashString("static_cast", this), new Integer(23));
    literals.put(new ANTLRHashString("const_cast", this), new Integer(9));
    literals.put(new ANTLRHashString("continue", this), new Integer(10));
    literals.put(new ANTLRHashString("do", this), new Integer(12));
    literals.put(new ANTLRHashString("typedef", this), new Integer(27));
    literals.put(new ANTLRHashString("namespace", this), new Integer(18));
    literals.put(new ANTLRHashString("enum", this), new Integer(14));
    literals.put(new ANTLRHashString("while", this), new Integer(29));
    literals.put(new ANTLRHashString("const", this), new Integer(8));
    literals.put(new ANTLRHashString("break", this), new Integer(6));
    literals.put(new ANTLRHashString("return", this), new Integer(20));
    literals.put(new ANTLRHashString("if", this), new Integer(17));
    literals.put(new ANTLRHashString("default", this), new Integer(11));
    literals.put(new ANTLRHashString("struct", this), new Integer(24));
    literals.put(new ANTLRHashString("using", this), new Integer(19));
    literals.put(new ANTLRHashString("else", this), new Integer(13));
    caseSensitiveLiterals = true;
    setCaseSensitive(true);
}

public Token nextToken() throws TokenStreamException {
    Token theRetToken=null;
    tryAgain:
    for (;;) {

```

```

Token _token = null;
int _ttype = Token.INVALID_TYPE;
resetText();
try { // for char stream error handling
  try { // for lexical error handling
    switch ( LA(1) ) {
      case '[':
      {
        mLBRACKET(true);
        theRetToken=_returnToken;
        break;
      }
      case ']':
      {
        mRBRACKET(true);
        theRetToken=_returnToken;
        break;
      }
      case '(':
      {
        mLPAREN(true);
        theRetToken=_returnToken;
        break;
      }
      case ')':
      {
        mRPAREN(true);
        theRetToken=_returnToken;
        break;
      }
      case '{':
      {
        mLBRACE(true);
        theRetToken=_returnToken;
        break;
      }
      case '}':
      {
        mRBRACE(true);
        theRetToken=_returnToken;
        break;
      }
      case '~':
      {
        mTILDE(true);
        theRetToken=_returnToken;
        break;
      }
      case '?':
      {
        mQUESTION(true);
        theRetToken=_returnToken;
        break;
      }
      case ';':
      {
        mSEMI(true);
        theRetToken=_returnToken;
        break;
      }
      case ',':
      {
        mCOMMA(true);
        theRetToken=_returnToken;
        break;
      }
      case '\ t': case '\ n': case '\000c': case '\ r':
      case ' ':
      {
        mWS(true);
        theRetToken=_returnToken;
        break;
      }
      case 'A': case 'B': case 'C': case 'D':

```

```

case 'E': case 'F': case 'G': case 'H':
case 'I': case 'J': case 'K': case 'L':
case 'M': case 'N': case 'O': case 'P':
case 'Q': case 'R': case 'S': case 'T':
case 'U': case 'V': case 'W': case 'X':
case 'Y': case 'Z': case '_': case 'a':
case 'b': case 'c': case 'd': case 'e':
case 'f': case 'g': case 'h': case 'i':
case 'j': case 'k': case 'l': case 'm':
case 'n': case 'o': case 'p': case 'q':
case 'r': case 's': case 't': case 'u':
case 'v': case 'w': case 'x': case 'y':
case 'z':
{
  mIDENTIFIER(true);
  theRetToken=_returnToken;
  break;
}
case '.': case '0': case '1': case '2':
case '3': case '4': case '5': case '6':
case '7': case '8': case '9':
{
  mINTEGER_CONSTANT(true);
  theRetToken=_returnToken;
  break;
}
case '`':
{
  mCHAR_CONSTANT(true);
  theRetToken=_returnToken;
  break;
}
case '"':
{
  mSTRING_LITERAL(true);
  theRetToken=_returnToken;
  break;
}
default:
  if ((LA(1)=='<') && (LA(2)=='<') && (LA(3)=='=')) {
    mLSHIFT_ASSIGN(true);
    theRetToken=_returnToken;
  }
  else if ((LA(1)=='>') && (LA(2)=='>') && (LA(3)=='=')) {
    mRSHIFT_ASSIGN(true);
    theRetToken=_returnToken;
  }
  else if ((LA(1)=='+' && (LA(2)=='+')) {
    mINCR(true);
    theRetToken=_returnToken;
  }
  else if ((LA(1)=='-' && (LA(2)=='-')) {
    mDECR(true);
    theRetToken=_returnToken;
  }
  else if ((LA(1)=='<' && (LA(2)=='<') && (true)) {
    mLSHIFT(true);
    theRetToken=_returnToken;
  }
  else if ((LA(1)=='>' && (LA(2)=='>') && (true)) {
    mRSHIFT(true);
    theRetToken=_returnToken;
  }
  else if ((LA(1)=='<' && (LA(2)=='=')) {
    mLE(true);
    theRetToken=_returnToken;
  }
  else if ((LA(1)=='>' && (LA(2)=='=')) {
    mGE(true);
    theRetToken=_returnToken;
  }
  else if ((LA(1)=='=' && (LA(2)=='=')) {
    mEQ(true);
    theRetToken=_returnToken;
  }

```

```

}
else if ((LA(1)=='!' && (LA(2)=='=')) {
    mNE(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='&' && (LA(2)=='&')) {
    mLAND(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='|' && (LA(2)=='|')) {
    mLOR(true);
    theRetToken=_returnToken;
}
else if ((LA(1)==':' && (LA(2)=='::')) {
    mCOLON_COLON(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='*' && (LA(2)=='=')) {
    mSTAR_ASSIGN(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='/' && (LA(2)=='=')) {
    mSLASH_ASSIGN(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='%' && (LA(2)=='=')) {
    mPERCENT_ASSIGN(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='+' && (LA(2)=='=')) {
    mPLUS_ASSIGN(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='-' && (LA(2)=='=')) {
    mMINUS_ASSIGN(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='&' && (LA(2)=='=')) {
    mAMPERSAND_ASSIGN(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='^' && (LA(2)=='=')) {
    mCARET_ASSIGN(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='|' && (LA(2)=='=')) {
    mPIPE_ASSIGN(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='/' && (LA(2)=='/')) {
    mSL_COMMENT(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='/' && (LA(2)=='*')) {
    mML_COMMENT(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='#' && (_tokenSet_0.member(LA(2)))) {
    mPOUND_INCLUDE_DIRECTIVE(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='&' && (true)) {
    mAMPERSAND(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='*' && (true)) {
    mSTAR(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='+' && (true)) {
    mPLUS(true);
    theRetToken=_returnToken;
}
else if ((LA(1)=='-' && (true)) {

```

```

        mMINUS(true);
        theRetToken=_returnToken;
    }
    else if ((LA(1)=='!') && (true)) {
        mBANG(true);
        theRetToken=_returnToken;
    }
    else if ((LA(1)=='/') && (true)) {
        mSLASH(true);
        theRetToken=_returnToken;
    }
    else if ((LA(1)=='%') && (true)) {
        mPERCENT(true);
        theRetToken=_returnToken;
    }
    else if ((LA(1)=='<') && (true)) {
        mLT(true);
        theRetToken=_returnToken;
    }
    else if ((LA(1)=='>') && (true)) {
        mGT(true);
        theRetToken=_returnToken;
    }
    else if ((LA(1)=='^') && (true)) {
        mCARET(true);
        theRetToken=_returnToken;
    }
    else if ((LA(1)=='|') && (true)) {
        mPIPE(true);
        theRetToken=_returnToken;
    }
    else if ((LA(1)==':') && (true)) {
        mCOLON(true);
        theRetToken=_returnToken;
    }
    else if ((LA(1)=='=') && (true)) {
        mASSIGN(true);
        theRetToken=_returnToken;
    }
    else if ((LA(1)=='#') && (true)) {
        mPOUND(true);
        theRetToken=_returnToken;
    }
    }
    else {
        if (LA(1)==EOF_CHAR) {uponEOF(); _returnToken = makeToken(Token.EOF_TYPE);}
        else {throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());}
    }
    }
    if ( _returnToken==null ) continue tryAgain; // found SKIP token
    _ttype = _returnToken.getType();
    _ttype = testLiteralsTable(_ttype);
    _returnToken.setType(_ttype);
    return _returnToken;
}
catch (RecognitionException e) {
    throw new TokenStreamRecognitionException(e);
}
}
catch (CharStreamException cse) {
    if ( cse instanceof CharStreamIOException ) {
        throw new TokenStreamIOException(((CharStreamIOException)cse).io);
    }
    else {
        throw new TokenStreamException(cse.getMessage());
    }
}
}
}

public final void mLBACKET(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = LBACKET;
    int _saveIndex;

```

```

    match('[');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mRBRACKET(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = RBRACKET;
    int _saveIndex;

    match(']');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mLAREN(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = LPAREN;
    int _saveIndex;

    match('(');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mRPAREN(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = RPAREN;
    int _saveIndex;

    match(')');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mLBRACE(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = LBRACE;
    int _saveIndex;

    match('{');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mRBRACE(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = RBRACE;
    int _saveIndex;

    match('}');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
}

```

```

    }
    _returnToken = _token;
}

public final void mINCR(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = INCR;
    int _saveIndex;

    match("++");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mDECR(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = DECR;
    int _saveIndex;

    match("--");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mAMPERSAND(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = AMPERSAND;
    int _saveIndex;

    match('&');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mSTAR(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = STAR;
    int _saveIndex;

    match('*');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mPLUS(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = PLUS;
    int _saveIndex;

    match('+');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

```

```

public final void mMINUS(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = MINUS;
    int _saveIndex;

    match('-');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mTILDE(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = TILDE;
    int _saveIndex;

    match('~');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mBANG(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = BANG;
    int _saveIndex;

    match('!');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mSLASH(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = SLASH;
    int _saveIndex;

    match('/');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mPERCENT(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = PERCENT;
    int _saveIndex;

    match('%');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mLSHIFT(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = LSHIFT;

```

```

    int _saveIndex;

    match("<<");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mRSHIFT(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = RSHIFT;
    int _saveIndex;

    match(">>");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mLT(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = LT;
    int _saveIndex;

    match('<');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mGT(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = GT;
    int _saveIndex;

    match('>');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mLE(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = LE;
    int _saveIndex;

    match("<=");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mGE(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = GE;
    int _saveIndex;

    match(">=");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {

```

```

        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mEQ(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = EQ;
    int _saveIndex;

    match("==");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mNE(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = NE;
    int _saveIndex;

    match("!=");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mCARET(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = CARET;
    int _saveIndex;

    match('^');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mPIPE(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = PIPE;
    int _saveIndex;

    match('|');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mLAND(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = LAND;
    int _saveIndex;

    match("&&");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

```

```

}

public final void mLOR(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = LOR;
    int _saveIndex;

    match("||");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mQUESTION(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = QUESTION;
    int _saveIndex;

    match('?');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mCOLON(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = COLON;
    int _saveIndex;

    match(':');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mCOLON_COLON(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = COLON_COLON;
    int _saveIndex;

    match("::");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mSEMI(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = SEMI;
    int _saveIndex;

    match(';');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mASSIGN(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {

```

```

int _ttype; Token _token=null; int _begin=text.length();
_ttype = ASSIGN;
int _saveIndex;

match('=');
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
    _token = makeToken(_ttype);
    _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mSTAR_ASSIGN(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = STAR_ASSIGN;
    int _saveIndex;

    match("*");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mSLASH_ASSIGN(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = SLASH_ASSIGN;
    int _saveIndex;

    match("/");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mPERCENT_ASSIGN(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = PERCENT_ASSIGN;
    int _saveIndex;

    match("%");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mPLUS_ASSIGN(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = PLUS_ASSIGN;
    int _saveIndex;

    match("+");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mMINUS_ASSIGN(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = MINUS_ASSIGN;
    int _saveIndex;

```

```

    match("-=");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mCOMMA(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = COMMA;
    int _saveIndex;

    match(',');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mPOUND(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = POUND;
    int _saveIndex;

    match('#');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

/** These currently seem to be unused. */
public final void mLSHIFT_ASSIGN(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = LSHIFT_ASSIGN;
    int _saveIndex;

    match("<<=");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mRSHIFT_ASSIGN(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = RSHIFT_ASSIGN;
    int _saveIndex;

    match(">>=");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mAMPERSAND_ASSIGN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = AMPERSAND_ASSIGN;
    int _saveIndex;

    match("&=");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);

```

```

        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mCARET_ASSIGN(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = CARET_ASSIGN;
    int _saveIndex;

    match("~=");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mPIPE_ASSIGN(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = PIPE_ASSIGN;
    int _saveIndex;

    match("|=");
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mWS(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = WS;
    int _saveIndex;

    {
        switch ( LA(1) ) {
        case '\ t': case '\00c': case ' ':
            {
                mWHITESPACE(false);
                break;
            }
        case '\ n': case '\ r':
            {
                mNEW_LINE(false);
                break;
            }
        default:
            {
                throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
            }
        }
    }
    _ttype = Token.SKIP;
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

protected final void mWHITESPACE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = WHITESPACE;
    int _saveIndex;

    switch ( LA(1) ) {
    case ' ':
        {

```

```

        match(' ');
        break;
    }
    case '\ t':
    {
        match('\ t');
        break;
    }
    case '00c':
    {
        match('\ f');
        break;
    }
    default:
    {
        throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
    }
}
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
    _token = makeToken(_ttype);
    _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

protected final void mNEW_LINE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = NEW_LINE;
    int _saveIndex;

    if ((LA(1)=='\ r') && (LA(2)=='\ n')) {
        match("\ r\ n");
        newline();
    }
    else if ((LA(1)=='\ r') && (true)) {
        match('\ r');
        newline();
    }
    else if ((LA(1)=='\ n')) {
        match('\ n');
        newline();
    }
    else {
        throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
    }

    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mSL_COMMENT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = SL_COMMENT;
    int _saveIndex;

    match("//");
    {
    _loop53:
    do {
        if ((_tokenSet_1.member(LA(1)))) {
            {
            match(_tokenSet_1);
            }
        }
        else {
            break _loop53;
        }
    }
    } while (true);
}

```

```

}
{
switch ( LA(1)) {
case '\ n':
{
match('\ n');
break;
}
case '\ r':
{
match('\ r');
{
if ((LA(1)=='\ n')) {
match('\ n');
}
else {
}
}
break;
}
default:
{
}
}
}
_ttype = Token.SKIP; newline();
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
_token = makeToken(_ttype);
_token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mML_COMMENT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
int _ttype; Token _token=null; int _begin=text.length();
_ttype = ML_COMMENT;
int _saveIndex;

match("/ *");
{
_loop59:
do {
if (((LA(1)=='*') && ((LA(2) >= '0003' && LA(2) <= '00ff')) &&
((LA(3) >= '0003' && LA(3) <= '00ff')) && ( LA(2)!='/' ) ) {
match('*');
}
else if ((LA(1)=='\ r') && (LA(2)=='\ n')
&& ((LA(3) >= '0003' && LA(3) <= '00ff')) {
match('\ r');
match('\ n');
newline();
}
else if ((LA(1)=='\ r') && ((LA(2) >= '0003' &&
LA(2) <= '00ff')) && ((LA(3) >= '0003' && LA(3) <= '00ff')) {
match('\ r');
newline();
}
else if ((LA(1)=='\ n')) {
match('\ n');
newline();
}
else if ((_tokenSet_2.member(LA(1)))) {
{
match(_tokenSet_2);
}
}
else {
break _loop59;
}
} while (true);
}

```

```

match("*/");
_ttype = Token.SKIP;
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
    _token = makeToken(_ttype);
    _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

public final void mIDENTIFIER(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = IDENTIFIER;
    int _saveIndex;

    mIDENTIFIER_NONDIGIT(false);
    {
    _loop62:
    do {
        switch ( LA(1) ) {
            case 'A': case 'B': case 'C': case 'D':
            case 'E': case 'F': case 'G': case 'H':
            case 'I': case 'J': case 'K': case 'L':
            case 'M': case 'N': case 'O': case 'P':
            case 'Q': case 'R': case 'S': case 'T':
            case 'U': case 'V': case 'W': case 'X':
            case 'Y': case 'Z': case '_': case 'a':
            case 'b': case 'c': case 'd': case 'e':
            case 'f': case 'g': case 'h': case 'i':
            case 'j': case 'k': case 'l': case 'm':
            case 'n': case 'o': case 'p': case 'q':
            case 'r': case 's': case 't': case 'u':
            case 'v': case 'w': case 'x': case 'y':
            case 'z':
            {
                mIDENTIFIER_NONDIGIT(false);
                break;
            }
            case '0': case '1': case '2': case '3':
            case '4': case '5': case '6': case '7':
            case '8': case '9':
            {
                mDIGIT(false);
                break;
            }
            default:
            {
                break _loop62;
            }
        }
    } while (true);
    }
    if(libraryManager.getTypeManager().isTypeName(new String(text.getBuffer(),
_begin,text.length()-_begin)) _ttype = TYPENAME;
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

protected final void mIDENTIFIER_NONDIGIT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = IDENTIFIER_NONDIGIT;
    int _saveIndex;

    switch ( LA(1) ) {
        case 'A': case 'B': case 'C': case 'D':
        case 'E': case 'F': case 'G': case 'H':
        case 'I': case 'J': case 'K': case 'L':
        case 'M': case 'N': case 'O': case 'P':
        case 'Q': case 'R': case 'S': case 'T':
        case 'U': case 'V': case 'W': case 'X':

```

```

case 'Y': case 'Z':
{
    matchRange('A','Z');
    break;
}
case 'a': case 'b': case 'c': case 'd':
case 'e': case 'f': case 'g': case 'h':
case 'i': case 'j': case 'k': case 'l':
case 'm': case 'n': case 'o': case 'p':
case 'q': case 'r': case 's': case 't':
case 'u': case 'v': case 'w': case 'x':
case 'y': case 'z':
{
    matchRange('a','z');
    break;
}
case '_':
{
    match('_');
    break;
}
default:
{
    throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
}
}
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
    _token = makeToken(_ttype);
    _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

protected final void mDIGIT(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = DIGIT;
    int _saveIndex;

    matchRange('0','9');
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mINTEGER_CONSTANT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = INTEGER_CONSTANT;
    int _saveIndex;
    boolean isDouble = true, isLong = false;

    switch ( LA(1) ) {
    case '.':
    {
        match('.');
        _ttype = DOT;
        {
            if (((LA(1) >= '0' && LA(1) <= '9')) ) {
                {
                    int _cnt68=0;
                    _loop68:
                    do {
                        if (((LA(1) >= '0' && LA(1) <= '9')) ) {
                            mDIGIT(false);
                        }
                        else {
                            if ( _cnt68>=1 ) { break _loop68; } else {throw new NoViableAltForCharException((char)LA(1),
                                getFilename(), getLine());}
                        }
                    } while (true);
                    _cnt68++;
                }
            }
        }
    }
}

```

```

    } while (true);
    }
    {
    if ((LA(1)=='E' || LA(1)=='e') {
        mEXPONENT(false);
    }
    else {
    }

    }
    {
    if ((LA(1)=='F' || LA(1)=='f') {
        mFLOATING_SUFFIX(false);
        isDouble = false;
    }
    else {
    }

    }

    CToken ct = new CToken(FLOATING_CONSTANT, new String(text.getBuffer(),_begin,
text.length()-_begin));

    try {
        if(isDouble)
            ct.setValue(Double.valueOf(new String(text.getBuffer(),_begin,text.length()-_begin)));
        else
            ct.setValue(Float.valueOf(new String(text.getBuffer(),_begin,text.length()-_begin)));
    } catch(NumberFormatException ex) { }
    _token = ct;

}
else {
}

}
break;
}
case '0': case '1': case '2': case '3':
case '4': case '5': case '6': case '7':
case '8': case '9':
{
{
int _cnt72=0;
_loop72:
do {
if (((LA(1) >= '0' && LA(1) <= '9')) {
    mDIGIT(false);
}
else {
if ( _cnt72>=1 ) { break _loop72; } else {throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine());}
}

_cnt72++;
} while (true);
}

CToken ict = new CToken(INTEGER_CONSTANT, new String(text.getBuffer(),
_begin,text.length()-_begin));

try {
    ict.setValue(Integer.valueOf(new String(text.getBuffer(),
_begin,text.length()-_begin)));
} catch(NumberFormatException ex) { }
_token = ict;

{
switch ( LA(1)) {
case 'L': case 'l':
{
{
_saveIndex=text.length();
mINTEGER_SUFFIX(false);

```



```

        ct.setValue(Double.valueOf(new String(text.getBuffer(),_begin,text.length()-_begin)));
    else
        ct.setValue(Float.valueOf(new String(text.getBuffer(),_begin,text.length()-_begin)));
    } catch(NumberFormatException ex) { }
    _token = ct;

    }
    break;
}
default:
{
}
}
}
break;
}
default:
{
    throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
}
}
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
    _token = makeToken(_ttype);
    _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

protected final void mEXPONENT(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = EXPONENT;
    int _saveIndex;

    {
    switch ( LA(1)) {
    case 'e':
    {
        match('e');
        break;
    }
    case 'E':
    {
        match('E');
        break;
    }
    default:
    {
        throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
    }
    }
    }
    {
    switch ( LA(1)) {
    case '+':
    {
        mPLUS(false);
        break;
    }
    case '-':
    {
        mMINUS(false);
        break;
    }
    case '0': case '1': case '2': case '3':
    case '4': case '5': case '6': case '7':
    case '8': case '9':
    {
        break;
    }
    default:
    {
        throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
    }
    }
    }
}

```

```

    }
  }
  {
  int _cnt86=0;
  _loop86:
  do {
    if (((LA(1) >= '0' && LA(1) <= '9')) {
      mDIGIT(false);
    }
    else {
      if ( _cnt86>=1 ) { break _loop86; } else {throw new NoViableAltForCharException((char)LA(1),
        getFilename(), getLine());}
    }
  }

  _cnt86++;
} while (true);
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
  _token = makeToken(_ttype);
  _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

protected final void mFLOATING_SUFFIX(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
  int _ttype; Token _token=null; int _begin=text.length();
  _ttype = FLOATING_SUFFIX;
  int _saveIndex;

  switch ( LA(1) ) {
  case 'f':
    {
      match('f');
      break;
    }
  case 'F':
    {
      match('F');
      break;
    }
  default:
    {
      throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
    }
  }
  if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
    _token = makeToken(_ttype);
    _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
  }
  _returnToken = _token;
}

protected final void mINTEGER_SUFFIX(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
  int _ttype; Token _token=null; int _begin=text.length();
  _ttype = INTEGER_SUFFIX;
  int _saveIndex;

  switch ( LA(1) ) {
  case '1':
    {
      match('1');
      break;
    }
  case 'L':
    {
      match('L');
      break;
    }
  default:
    {
      throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
    }
  }
}

```

```

    }
    if ( !_createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mCHAR_CONSTANT(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = CHAR_CONSTANT;
    int _saveIndex;

    match('');
    {
    if ((LA(1)=='\\') {
        mESC(false);
    }
    else if ((_tokenSet_3.member(LA(1)))) {
        matchNot('');
    }
    else {
        throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
    }
    }
    match('');

    CToken ct = new CToken(CHAR_CONSTANT, new String(text.getBuffer(),_begin,text.length()-_begin));

    ct.setValue(new Character(EscapeTranslator.translateEscapeSequence(new String(text.getBuffer(),
    _begin,text.length()-_begin).charAt(1))));
    _token = ct;

    if ( !_createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

protected final void mESC(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = ESC;
    int _saveIndex;

    match('\\');
    {
    switch ( LA(1) ) {
    case 'n':
    {
        match('n');
        break;
    }
    case 'r':
    {
        match('r');
        break;
    }
    case 't':
    {
        match('t');
        break;
    }
    case 'b':
    {
        match('b');
        break;
    }
    case 'f':
    {
        match('f');

```

```

    break;
}
case '":
{
    match('"');
    break;
}
case '\':
{
    match('\\');
    break;
}
case '\\':
{
    match('\\\\');
    break;
}
case '0': case '1': case '2': case '3':
{
    {
        matchRange('0','3');
    }
    {
        if (((LA(1) >= '0' && LA(1) <= '7') && ((LA(2) >= '0003' && LA(2) <= '00ff')) && (true)) {
            {
                matchRange('0','7');
            }
            {
                if (((LA(1) >= '0' && LA(1) <= '7') && ((LA(2) >= '0003' && LA(2) <= '00ff')) && (true)) {
                    {
                        matchRange('0','7');
                    }
                }
            }
            else if (((LA(1) >= '0003' && LA(1) <= '00ff')) && (true) && (true)) {
            }
            else {
                throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
            }
        }
    }
}
else if (((LA(1) >= '0003' && LA(1) <= '00ff')) && (true) && (true)) {
}
else {
    throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
}
}
break;
}
case '4': case '5': case '6': case '7':
{
    {
        matchRange('4','7');
    }
    {
        if (((LA(1) >= '0' && LA(1) <= '9') && ((LA(2) >= '0003' && LA(2) <= '00ff')) && (true)) {
            {
                matchRange('0','9');
            }
        }
        else if (((LA(1) >= '0003' && LA(1) <= '00ff')) && (true) && (true)) {
        }
        else {
            throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
        }
    }
}
break;
}
default:
{
    throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
}
}

```

```

    }
  }
  if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
    _token = makeToken(_ttype);
    _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
  }
  _returnToken = _token;
}

public final void mSTRING_LITERAL(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
  int _ttype; Token _token=null; int _begin=text.length();
  _ttype = STRING_LITERAL;
  int _saveIndex;

  match('"');
  {
  _loop94:
  do {
    if ((LA(1)=='\\')) {
      mESC(false);
    }
    else if ((_tokenSet_4.member(LA(1)))) {
      {
      match(_tokenSet_4);
      }
    }
    else {
      break _loop94;
    }
  }
  } while (true);
  }
  match('"');

  CToken ct = new CToken(STRING_LITERAL, new String(text.getBuffer(),_begin,text.length()-_begin));

  ct.setValue(EscapeTranslator.translateEscapeSequence(new String(text.getBuffer(),_begin,
text.length()-_begin)));
  _token = ct;

  if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
    _token = makeToken(_ttype);
    _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
  }
  _returnToken = _token;
}

protected final void mHEADER_NAME(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
  int _ttype; Token _token=null; int _begin=text.length();
  _ttype = HEADER_NAME;
  int _saveIndex;

  switch ( LA(1) ) {
  case '<':
    {
      _saveIndex=text.length();
      mLT(false);
      text.setLength(_saveIndex);
      {
      _loop108:
      do {
        if ((_tokenSet_5.member(LA(1)))) {
          {
          match(_tokenSet_5);
          }
        }
        else {
          break _loop108;
        }
      }
      } while (true);
    }
  }
}

```

```

    _saveIndex=text.length();
    mGT(false);
    text.setLength(_saveIndex);

    CToken ct = new CToken(HEADER_NAME, new String(text.getBuffer(),_begin,text.length()-_begin));

    ct.setValue(new String(text.getBuffer(),_begin,text.length()-_begin));
    _token = ct;

    break;
}
case '"':
{
    _saveIndex=text.length();
    match('"');
    text.setLength(_saveIndex);
    {
        _loop111:
        do {
            if ((_tokenSet_6.member(LA(1)))) {
                {
                    match(_tokenSet_6);
                }
            }
            else {
                break _loop111;
            }
        } while (true);
    }
    _saveIndex=text.length();
    match('"');
    text.setLength(_saveIndex);

    CToken ct = new CToken(HEADER_NAME, new String(text.getBuffer(),_begin,text.length()-_begin));

    ct.setValue(new String(text.getBuffer(),_begin,text.length()-_begin));
    _token = ct;

    break;
}
default:
{
    throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
}
}
if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
    _token = makeToken(_ttype);
    _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

protected final void mPP_TOKENS(boolean _createToken) throws RecognitionException, CharStreamException,
TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = PP_TOKENS;
    int _saveIndex;

    {
        int _cnt114=0;
        _loop114:
        do {
            if ((LA(1)=='"'||LA(1)=='<') ) {
                mPREPROCESSING_TOKEN(false);
            }
            else {
                if ( _cnt114>=1 ) { break _loop114; } else {throw new NoViableAltForCharException((char)LA(1),
getFilename(), getLine());}
            }

            _cnt114++;
        } while (true);
    }
}

```

```

    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(text.getBuffer(), _begin, text.length()-_begin);
    }
    _returnToken = _token;
}

protected final void mPREPROCESSING_TOKEN(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = PREPROCESSING_TOKEN;
    int _saveIndex;

    mHEADER_NAME(false);
    if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
        _token = makeToken(_ttype);
        _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
    }
    _returnToken = _token;
}

public final void mPOUND_INCLUDE_DIRECTIVE(boolean _createToken) throws RecognitionException,
CharStreamException, TokenStreamException {
    int _ttype; Token _token=null; int _begin=text.length();
    _ttype = POUND_INCLUDE_DIRECTIVE;
    int _saveIndex;
    Token p=null;

    _saveIndex=text.length();
    mPOUND(false);
    text.setLength(_saveIndex);
    {
    _loop118:
    do {
        if ((LA(1)=='\ t' || LA(1)=='\00c' || LA(1)==' ') {
            mWHITESPACE(false);
        }
        else {
            break _loop118;
        }
    } while (true);
    }
    _saveIndex=text.length();
    match("include");
    text.setLength(_saveIndex);
    {
    _loop120:
    do {
        if ((LA(1)=='\ t' || LA(1)=='\00c' || LA(1)==' ') {
            mWHITESPACE(false);
        }
        else {
            break _loop120;
        }
    } while (true);
    }
    mPP_TOKENS(true);
    p=_returnToken;
    {
    _loop122:
    do {
        if ((LA(1)=='\ t' || LA(1)=='\00c' || LA(1)==' ') {
            mWHITESPACE(false);
        }
        else {
            break _loop122;
        }
    } while (true);
    }
    {
    if ((LA(1)=='/') && (LA(2)=='/')) {

```

```

        _saveIndex=text.length();
        mSL_COMMENT(false);
        text.setLength(_saveIndex);
    }
    else if ((LA(1)=='/') && (LA(2)=='*')) {
        _saveIndex=text.length();
        mML_COMMENT(false);
        text.setLength(_saveIndex);
        _saveIndex=text.length();
        mNEW_LINE(false);
        text.setLength(_saveIndex);
    }
    else if ((LA(1)=='\ n' || LA(1)=='\ r')) {
        _saveIndex=text.length();
        mNEW_LINE(false);
        text.setLength(_saveIndex);
    }
    else {
        throw new NoViableAltForCharException((char)LA(1), getFilename(), getLine());
    }
}

    try {
        libraryManager.loadLibrary(p.getText());
    }
    catch(NoSuchLibraryException ex) {
        reportError("There is no library named " + p.getText() + "",
            getLine() - 1, getColumn(), ErrorCodeTable.INVALID_INCLUDE_FILE);
    }
    catch(LibraryInitializationException ex) {
        reportError("Error initializing library " + p.getText() + "",
            getLine() - 1, getColumn(), ErrorCodeTable.ERROR_INITIALIZING_INCLUDE);
    }
}

if ( _createToken && _token==null && _ttype!=Token.SKIP ) {
    _token = makeToken(_ttype);
    _token.setText(new String(text.getBuffer(), _begin, text.length()-_begin));
}
_returnToken = _token;
}

private static final long _tokenSet_0_data_[] = { 4294971904L, 219902325552L, 0L, 0L, 0L };
public static final BitSet _tokenSet_0 = new BitSet(_tokenSet_0_data_);
private static final long _tokenSet_1_data_[] = { -9224L, -1L, -1L, -1L, 0L, 0L, 0L, 0L };
public static final BitSet _tokenSet_1 = new BitSet(_tokenSet_1_data_);
private static final long _tokenSet_2_data_[] = { -4398046520328L, -1L, -1L, -1L, 0L, 0L, 0L, 0L };
public static final BitSet _tokenSet_2 = new BitSet(_tokenSet_2_data_);
private static final long _tokenSet_3_data_[] = { -549755813896L, -268435457L, -1L, -1L, 0L, 0L, 0L, 0L };
public static final BitSet _tokenSet_3 = new BitSet(_tokenSet_3_data_);
private static final long _tokenSet_4_data_[] = { -17179869192L, -268435457L, -1L, -1L, 0L, 0L, 0L, 0L };
public static final BitSet _tokenSet_4 = new BitSet(_tokenSet_4_data_);
private static final long _tokenSet_5_data_[] = { -4611686018427397128L, -1L, -1L, -1L, 0L, 0L, 0L, 0L };
public static final BitSet _tokenSet_5 = new BitSet(_tokenSet_5_data_);
private static final long _tokenSet_6_data_[] = { -17179878408L, -1L, -1L, -1L, 0L, 0L, 0L, 0L };
public static final BitSet _tokenSet_6 = new BitSet(_tokenSet_6_data_);
}

```

J.3 ANTLR Input File Used to Create *CS1 Sandbox*'s Parser (CParser.g)

```

header {
package sandbox.compiler.frontend;

import java.util.*;
import sandbox.compiler.*;
import sandbox.client.ErrorCodeTable;
}

/**
 * Contains the grammar for the C Interpreter. The grammar (or a close
 * facsimile) is specified in docs/internal/grammar.html.
 */
class CParser extends Parser;

options {
k = 3;          // # tokens of lookahead
importVocab = CLexer;
exportVocab = CParser;
ASTLabelType = SandboxAST;
buildAST = true;
}

// "Imaginary" tokens. These are tokens used to build up the AST and
// simplify the grammar. They may not have any actual equivalent in
// the source code.
tokens {
ARGUMENTS;
ARRAY_DECLARATOR;
BLOCK_STATEMENT;
CALL;
CASE_GROUP;
CAST;
DECLARATION;
DECLARATOR;
ENUMERATOR_LIST;
EXPRESSION_STATEMENT;
FOR_INIT;
FOR_COND;
FOR_ITER;
FUNCTION;
FUNCTION_PROTOTYPE;
INCLUDE;
INITIALIZER;
PARAMETER;
PARAMETER_LIST;
POST_DECR;
POST_INCR;
PROGRAM;
SLIST;
STRUCT_DECLARATION;
STRUCT_MEMBERS;
SUBSCRIPT;
TYPE_MODIFIER;
TYPE_SIZEOF;
USING_DIRECTIVE;
}

{
/** The errors that are observed during the parse phase. */
private Vector errorVector = new Vector();

/** The C language features that are allowed/forbidden. */
private Restriction restriction;

/** Are we currently parsing a Typedef? */
boolean inTypedef = false;

/** The CLexer that we're using to get tokens from/add types to. */

```

```

private CLexer lexer;

/**
 * Returns a vector of each error seen during compilation.
 */
public Vector getErrorVector() {
    return(errorVector);
}

/**
 * Sets a reference to the current restriction object
 */
public void setRestriction(Restriction restr) {
    restriction = restr;
}

/**
 * ReportError method, supporting the error reporting similiar to what ANTLR does.
 * We added support for the line number and column via this method.
 */
public void reportError (java.lang.String errMsg, int code, int line, int col) {
    AntlrError error = new AntlrError ();
    error.setType("level");
    error.setErrorMessage(errMsg);
    error.setErrorCode(code);
    error.setLineNumber(line);
    error.setColumnNumber(col);
    errorVector.add(error);
}

/**
 * Overridden reportError method, supporting the error reporting via ANTLR
 */
public void reportError (int code, int line, int col) {
    AntlrError error = new AntlrError ();
    error.setType("level");
    error.setErrorCode (code);
    error.setLineNumber(line);
    error.setColumnNumber(col);
    errorVector.add(error);
}

/**
 * Overridden reportError method, supporting the error reporting via ANTLR
 */
public void reportError(RecognitionException ex) {
    String errorMsg;
    AntlrError error = new AntlrError();
    error.setType("recognition exception - parser error");

    // tack "-- parser" onto the raw message so that it can be classified later
    errorMsg = new String(ex.getMessage() + "-- parser");
    error.setErrorMessage(errorMsg);
    error.setLineNumber(ex.getLine());
    error.setColumnNumber(ex.getColumn());
    errorVector.add(error);
}

public void reportError(String s) {
    AntlrError error = new AntlrError();
    error.setType("error - parser error");
    error.setErrorMessage(s);
    errorVector.add(error);
}

public void reportWarning(String s) {
    AntlrError error = new AntlrError();

```

```

        error.setType("warning");
        error.setErrorMessage(s);
        errorVector.add(error);
    }

    /**
     * Set the lexer to use when adding types (necessary for typedef support).
     */
    public void setLexer(CLexer l) {
        lexer = l;
    }

    /**
     * Add the type 'type' to the lexer so that we can properly use it in
     * future variable declarations.
     */
    private void addType (String type) {
        // grab the lexer so we can add the type to the lexer.
        //lexer.getLibraryManager().getManager().registerTypeName(type);
    }
}

// eventually this will be the main item that gets parsed
program
: translationUnit EOF!
  { #program = #([PROGRAM, "PROGRAM"], #program); }
;

/** A.2.1 Expressions */
// mostly implemented -- typecasts, sizeof left

/* (6.5.1) primary-expr: */
primaryExpr
: IDENTIFIER
| constant
| STRING_LITERAL
| LPAREN! expression RPAREN!
;

/** (6.4.4) constant: */
constant
: INTEGER_CONSTANT
| FLOATING_CONSTANT
| CHAR_CONSTANT
| ( TRUE | FALSE )
;

/**
 * (6.5.2) postfix-expr:
 *
 * Postfix Expressions include postfix ++ and --, structure (union, class,
 * etc.) member access, function calls, array accesses, and the like.
 *
 * Array accesses are handled in arrayExpr.
 *
 * Shamelessly copied from the example grammar 'java.g'.
 */
postfixExpr
// constructor-style cast
: t:typeSpecifier LPAREN! e:expression RPAREN!
  { #postfixExpr = #([CAST, "CAST"], #postfixExpr); }
| STATIC_CAST LT t1:typeSpecifier GT LPAREN e1:expression RPAREN!
  { #postfixExpr = #([CAST, "CAST"], #postfixExpr); }
| CONST_CAST LT t2:typeSpecifier GT LPAREN e2:expression RPAREN!
  { #postfixExpr = #([CAST, "CAST"], #postfixExpr); }
| REINTERPRET_CAST LT t3:typeSpecifier GT LPAREN e3:expression RPAREN!
  { #postfixExpr = #([CAST, "CAST"], #postfixExpr); }
| i:primaryExpr // start with a primary

    ( // qualified id (id.id.id.id...) -- build the name
      DOT IDENTIFIER

```

```

// qualified id::id::id... -- build the name
// This is needed for nested classes, static data members,
// enumerations in class scope, etc...
| COLON_COLON IDENTIFIER

// method invocation
// The next line is not strictly proper; it allows x(3)(4) or
// x[2](4) which are not valid in Java. If this grammar were used
// to validate a Java program a semantic check would be needed, or
// this rule would get really ugly...
| LPAREN^
  a:(argumentExprList)? RPAREN!
  { #postfixExpr = #([CALL, "CALL"], i, a); }
)*

// possibly add on a post-increment or post-decrement.
// allows INC/DEC on too much, but semantics can check
( in:INCR^
{
  #in.setType(POST_INCR);
  if (!restriction.isEnabled(Restriction.POSTFIX) && in != null)
    reportError ("Postfix expressions", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
      in.getLine(), in.getColumn());
}

| de:DECR^
{
  #de.setType(POST_DECR);
  if (!restriction.isEnabled(Restriction.POSTFIX) && de != null)
    reportError ("Postfix expressions", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
      de.getLine(), de.getColumn());
}

| // nothing
)

;

memberAccess
: (primaryExpr (DOT^ IDENTIFIER)+ LPAREN) =>
  callMemberFunction
| primaryExpr (DOT^ IDENTIFIER)+
;

callMemberFunction
: primaryExpr (DOT^ IDENTIFIER)+ LPAREN! (argumentExprList)? RPAREN!
{ #callMemberFunction = #([CALL, "CALL"], #callMemberFunction); }
;

/**
 * Handle array access. If a '[' is observed, the next token must be
 * an expression of some form so that we know where to look in the array.
 */
arrayExpr
: (postfixExpr LBRACKET) => postfixExpr (LBRACKET! expression RBRACKET!)*
{ #arrayExpr = #([SUBSCRIPT, "SUBSCRIPT"], #arrayExpr); }
| postfixExpr
;

/** (6.5.2) argument-expr-list: list of the actual arguments to a function. */
argumentExprList
: assignmentExpr (COMMA! assignmentExpr)*
{ #argumentExprList = #([ARGUMENTS, "ARGUMENTS"], #argumentExprList); }
;

/** (6.5.3) unary-expr: */
unaryExpr
: (preInc:INCR^ | preDec:DECR^)? arrayExpr
{
  if (!restriction.isEnabled(Restriction.PREFIX))
    if (preInc != null)
      reportError ("Prefix expressions", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
        preInc.getLine(), preInc.getColumn());
    else if (preDec != null)
      reportError ("Prefix expressions", ErrorCodeTable.ARE_DISABLED_2_PARAMS,

```

```

        preDec.getLine(), preDec.getColumn());
    }
    | (PLUS^ | MINUS^ | TILDE^ | BANG^ ) castExpr
    ;

/** Handle <b>sizeof</b> expressions. This is an internal function. */
sizeofExpr
: SIZEOF^ unaryExpr
;

/** (6.5.4) cast-expr: */
castExpr
: (LPAREN LPAREN typeSpecifier => typecast2:LPAREN! LPAREN! typeSpecifier RPAREN! unaryExpr RPAREN!
{
#castExpr = #([CAST, "CAST"], #castExpr);
if (!restriction.isEnabled(Restriction.TYPECAST) && typecast2 != null)
reportError ("Explicit typecast statements", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
typecast2.getLine(), typecast2.getColumn());
}
| (LPAREN typeSpecifier => typecast:LPAREN! typeSpecifier RPAREN! unaryExpr
{
#castExpr = #([CAST, "CAST"], #castExpr);
if (!restriction.isEnabled(Restriction.TYPECAST) && typecast != null)
reportError ("Explicit typecast statements", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
typecast.getLine(), typecast.getColumn());
}
) unaryExpr
| sizeofExpr
;

/** (6.5.5) multiplicative-expr: */
multiplicativeExpr
: castExpr ((mult:STAR^ | div:SLASH^ | mod:PERCENT^ ) castExpr)*
{
if (!restriction.isEnabled(Restriction.MULTIPLICATION) && mult != null)
reportError ("Multiplication statements (*)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
mult.getLine(), mult.getColumn());

if (!restriction.isEnabled(Restriction.DIVISION) && div != null)
reportError ("Division statements (/)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
div.getLine(), div.getColumn());

if (!restriction.isEnabled(Restriction.MODULUS) && mod != null)
reportError ("Modulus statements (%)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
mod.getLine(), mod.getColumn());
}
;

/** (6.5.6) additive-expr: */
additiveExpr
: multiplicativeExpr ((addPlus:PLUS^ | addMinus:MINUS^ ) multiplicativeExpr)*
{
if (!restriction.isEnabled(Restriction.ADDITION) && addPlus != null)
reportError ("Addition statements (+)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
addPlus.getLine(), addPlus.getColumn());

if (!restriction.isEnabled(Restriction.SUBTRACTION) && addMinus != null)
reportError ("Subtraction statements (-)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
addMinus.getLine(), addMinus.getColumn());
}
;

/** (6.5.7) shift-expr: */
shiftExpr
: additiveExpr ((lshift:LSHIFT^ | rshift:RSHIFT^ ) additiveExpr)*
{
if (!restriction.isEnabled(Restriction.LEFT) && lshift != null)
reportError ("Left shift statements (<<)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
lshift.getLine(), lshift.getColumn());

if (!restriction.isEnabled(Restriction.RIGHT) && rshift != null)
reportError ("Right shift statements (>>)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
rshift.getLine(), rshift.getColumn());
}
;

```

```

;

/** (6.5.8) relational-expr: */
relationalExpr
: shiftExpr ((lt:LT^ | gt:GT^ | ltet:LE^ | gtet:GE^ ) shiftExpr)*
{
    if (!restriction.isEnabled(Restriction.LESS_THAN) && lt != null)
        reportError ("Less-than comparison statements (<)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
            lt.getLine(), lt.getColumn());

    if (!restriction.isEnabled(Restriction.LESS_THAN_EQTO) && ltet != null)
        reportError ("Less-than-or-equal-to comparison statements (<=)",
            ErrorCodeTable.ARE_DISABLED_4_PARAMS, ltet.getLine(), ltet.getColumn());

    if (!restriction.isEnabled(Restriction.GREATER_THAN) && gt != null)
        reportError ("Greater-than comparison statements (>)",
            ErrorCodeTable.ARE_DISABLED_4_PARAMS, gt.getLine(), gt.getColumn());

    if (!restriction.isEnabled(Restriction.GREATER_THAN_EQTO) && gtet != null)
        reportError ("Greater-than-or-equal-to comparison statements (>=)",
            ErrorCodeTable.ARE_DISABLED_4_PARAMS, gtet.getLine(), gtet.getColumn());
}
;

/** (6.5.9) equality-expr: */
equalityExpr
: relationalExpr ((eq:EQ^ | ne:NE^ ) relationalExpr)*
{
    if (!restriction.isEnabled(Restriction.EQUALITY) && eq != null)
        reportError ("Equality comparison statements (==)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
            eq.getLine(), eq.getColumn());

    if (!restriction.isEnabled(Restriction.INEQUALITY) && ne != null)
        reportError ("Inequality comparison statements (!=)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
            ne.getLine(), ne.getColumn());
}
;

/** (6.5.13) logical-AND-expr: */
logicalAndExpr
: equalityExpr (land:LAND^ equalityExpr)*
{
    if (!restriction.isEnabled(Restriction.AND) && land != null)
        reportError ("Logical AND statements (&&)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
            land.getLine(), land.getColumn());
}
;

/** (6.5.14) logical-OR-expr: */
logicalOrExpr
: logicalAndExpr (lor:LOR^ logicalAndExpr)*
{
    if (!restriction.isEnabled(Restriction.OR) && lor != null)
        reportError ("Logical OR statements (||)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
            lor.getLine(), lor.getColumn());
}
;

/** (6.5.16) assignment-expr: */
assignmentExpr
: (unaryExpr (ASSIGN | STAR_ASSIGN | SLASH_ASSIGN | PERCENT_ASSIGN | PLUS_ASSIGN | MINUS_ASSIGN) =>
    unaryExpr (assign:ASSIGN^ | star:STAR_ASSIGN^ | slash:SLASH_ASSIGN^ | mod:PERCENT_ASSIGN^
    | add:PLUS_ASSIGN^ | sub:MINUS_ASSIGN^ ) assignmentExpr
    {
        if (!restriction.isEnabled(Restriction.ASSIGNMENT) && assign != null)
            reportError ("Assignment statements (=)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
                assign.getLine(), assign.getColumn());

        if (!restriction.isEnabled(Restriction.MULT_ASSIGN) && star != null)
            reportError ("Multiplication/assignment statements (*=)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
                star.getLine(), star.getColumn());

        if (!restriction.isEnabled(Restriction.DIV_ASSIGN) && slash != null)
            reportError ("Division/assignment statements (/=)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
                slash.getLine(), slash.getColumn());
    }
)

```

```

    if (!restriction.isEnabled(Restriction.MOD_ASSIGN) && mod != null)
        reportError ("Modulus/assignment statements (%=)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
            mod.getLine(), mod.getColumn());

    if (!restriction.isEnabled(Restriction.ADD_ASSIGN) && add != null)
        reportError ("Addition/assignment statements (+=)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
            add.getLine(), add.getColumn());

    if (!restriction.isEnabled(Restriction.SUB_ASSIGN) && sub != null)
        reportError ("Subtraction/assignment statements (-=)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
            sub.getLine(), sub.getColumn());
    }
| logicalOrExpr
;

/** (6.5.17) expression: */
expression
: assignmentExpr (COMMA^ assignmentExpr)*
;

/** (6.6 ) constant-expr */
constantExpr
: logicalOrExpr
;

/* A.2.2 Declarations */
/* Mostly implemented -- type support lacking (typedefs, enums) */

/**
 * (6.7) declaration: declare a variable/function.
 *
 * Note: This must be kept in sync with ‘blockItem’.
 */
declaration
: blockDeclaration SEMI!
  { #declaration = #[[DECLARATION, "DECLARATION"], #declaration]; }
| (declarationSpecifier)+ initDeclaratorList SEMI!
  { #declaration = #[[DECLARATION, "DECLARATION"], #declaration]; }

// typedef should be in declarationSpecifier -> storageClassSpecifier, but
// we need pre- and post-handling for ‘inTypedef’, which requires that the
// handling be done here.
| typedef:TYPEDEF
  { inTypedef = true; }
  (declarationSpecifier)+ initDeclaratorList SEMI!
  {
    if (!restriction.isEnabled(Restriction.TYPEDEF) && typedef != null)
        reportError ("Typedef statements", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
            typedef.getLine(), typedef.getColumn());
    #declaration = #[[DECLARATION, "DECLARATION"], #declaration];
    inTypedef = false;
  }
;

blockDeclaration
: usingDirective
;

usingDirective
: USING! NAMESPACE! nestedNameSpecifier
  { #usingDirective = #[[USING_DIRECTIVE, "USING_DIRECTIVE"], #usingDirective]; }
;

nestedNameSpecifier
: (IDENTIFIER)*
;

/**
 * (6.7) declaration-specifiers: modifiers for a declaration.
 *
 * The modifiers can change the type of the declaration (e.g. ‘typedef’
 * declares a new type), the scope of a declaration (e.g. ‘extern’,
 * ‘static’, ‘auto’, ‘register’).
 */

```

```

* The type of a declaration is also specified here (e.g. 'int').
*/
declarationSpecifier
  : (typeModifiers)? typeSpecifier (ptrOperator)?
  ;

ptrOperator
  : AMPERSAND
  ;

typeModifiers
  : (typeModifier)+
  { #typeModifiers = #[TYPE_MODIFIER, "TYPE_MODIFIER"], #typeModifiers; }
  ;

typeModifier
  : c:CONST
  {
    if (!restriction.isEnabled(Restriction.CONST) && c != null)
      reportError ("Constant identifiers", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
        c.getLine(), c.getColumn());
  }
  ;

/**
 * (6.7) init-declarator-list: comma-separated list of the names of
 * variables to declare.
 */
initDeclaratorList
  : (initDeclarator (COMMA! initDeclarator)*)?
  ;

/** (6.7) init-declarator: allows '=' to be used in variable declarations. */
initDeclarator
  : d:declarator (ASSIGN! initializer)?
  { if (inTypedef)
    addType (#d.getText());
    #initDeclarator = #[DECLARATOR, "DECLARATOR"], #initDeclarator; }
  ;

/**
 * (6.7.2) type-specifier: types that can be used in declarations.
 *
 */
typeSpecifier
  : TYPENAME
  | structSpecifier
  | enumSpecifier
  ;

/**
 * (6.7.2.1) struct-or-union-specifier: structure declarations.
 *
 * We're not implementing unions (yet).
 *
 * There's too much duplicated code here. However, it's needed in order to
 * allow two-token lookahead, which is needed to prevent the decl-struct
 * issue.
 */
structSpecifier
  // anonymous structures, e.g. 'struct { int i ; } foo;'
  : STRUCT LBRACE! structDeclarationList RBRACE!
  { #structSpecifier = #[STRUCT_DECLARATION, "STRUCT_DECLARATION"], #structSpecifier; }

  // structure declarations, e.g. 'struct foo { int bar ; } ;'
  // Note: We're following C++ name rules, instead of C rules, as we add the
  // type to our type table as if it were a typedef. Thus, the following is
  // allowed:
  //
  //   struct foo { int bar };
  //   bar b;
  //
  // This isn't proper C, as in C, structures have their own namespace.
  | STRUCT i:IDENTIFIER

```

```

    { addType (#i.getText());
      if (!restriction.isEnabled(Restriction.STRUCT) && i != null)
        reportError ("Creating user-defined structs", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
                    i.getLine(), i.getColumn());
    }

    // If this is missing, then it's a forward declaration
    ( LBRACE! structDeclarationList RBRACE! )?
    { #structSpecifier = #([STRUCT_DECLARATION, "STRUCT_DECLARATION"], #structSpecifier); }

    // when trying to declare a new variable, such as with the C name rules, e.g:
    //   struct foo { int bar; };
    //   struct foo f;
    //   f.bar = 42;
    | STRUCT TYPENAME

    // if this is present, then it's the actual declaration to a previous
    // forward-declaration.
    ( LBRACE! structDeclarationList RBRACE! )?
    { #structSpecifier = #([STRUCT_DECLARATION, "STRUCT_DECLARATION"], #structSpecifier); }
;

/** (6.7.2.1) struct-declaration-list: declare the members of a struct. */
structDeclarationList
: (structDeclaration)*
  { #structDeclarationList = #([STRUCT_MEMBERS, "STRUCT_MEMBERS"], #structDeclarationList); }
;

/**
 * (6.7.2.1) struct-declaration: declare an individual member of a structure
 */
structDeclaration
: specifierQualifierList structDeclaratorList SEMI!
;

/** (6.7.2.1) specifier-qualifier-list: */
specifierQualifierList
: (typeSpecifier | CONST)+
;

/** (6.7.2.1) struct-declaration-list: declares names of members of a struct. */
structDeclaratorList
: ( structDeclarator (COMMA! structDeclarator)* )?
;

/**
 * (6.7.2.1) struct-declarator: declares a single member.
 *
 * Also used to declare bitfields (unimplemented).
 */
structDeclarator
: declarator
;

/** (6.7.2.2) enum-specifier: the declaration format for enumerations. */
enumSpecifier
: e:ENUM (IDENTIFIER)? LBRACE! enumeratorList RBRACE!
  {
    if (!restriction.isEnabled(Restriction.ENUM) && e != null)
      reportError ("Using enumerated types", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
                  e.getLine(), e.getColumn());
  }
;

/** (6.7.2.2) enumerator-list: list of the "members" of an enumeration. */
enumeratorList
: enumerator (COMMA! enumerator)*
  { #enumeratorList = #([ENUMERATOR_LIST, "ENUMERATOR_LIST"], #enumeratorList); }
;

/**
 * (6.7.2.2) enumerator: actual "member(s)" of an enumeration.
 *
 * Consists of either a name, or a name and a value:

```

```

*
* enum foo
* {
*   bar,    // name
*   baz = 1 // name & value
* }
*/
enumerator
: IDENTIFIER (ASSIGN! INTEGER_CONSTANT)?
  { #enumerator = #([DECLARATOR, "DECLARATOR"], #enumerator); }
;

/** (6.7.5) declarator: is direct-declarator; pointers aren't implemented */
declarator
: (IDENTIFIER LPAREN) => IDENTIFIER LPAREN! parameterList RPAREN!
  | IDENTIFIER
  | IDENTIFIER (arrayDesignator)+
  { #declarator = #([ARRAY_DECLARATOR, "ARRAY_DECLARATOR"], #declarator); }
;

/** (6.7.5) parameter-list: list of parameters to a function. */
parameterList
: parameterDeclarations
  { #parameterList = #([PARAMETER_LIST, "PARAMETER_LIST"], #parameterList); }
;

/** list of parameters to a function. */
parameterDeclarations
: parameterDeclaration (COMMA! parameterDeclaration)*
;

/** (6.7.5) parameter-declaration: parameters to a function. */
parameterDeclaration
: (declarationSpecifier)+ (declarator)?
  { #parameterDeclaration = #([PARAMETER, "PARAMETER"], #parameterDeclaration); }
;

/** (6.7.5) identifier-list: list of identifiers (as used in function calls) */
identifierList
: IDENTIFIER (COMMA! IDENTIFIER)*
;

/** (6.7.6) type-name: allows declaring the name of a type. */
typeName
: specifierQualifierList
;

/** (6.7.7) typedef-name: introduces a new type. */
typedefName
: IDENTIFIER
;

/** (6.7.8) initializer: used to initialize variables. */
initializer
: assignmentExpr
  | LBRACE! initializerList (COMMA!)? RBRACE!
  { #initializer = #([INITIALIZER, "INITIALIZER"], #initializer); }
;

/**
* (6.7.8) initializer-list: used when initializing variables of structure type.
*
* For example:
* <pre>
* struct foo { int n; };
* struct foo f =
*   { // begin initializer-list
*     1 // end initializer-list
*   };
* </pre>
*/
initializerList
: (designation)? initializer (COMMA! (designation)? initializer)*
;

```

```

/**
 * (6.7.8) designation: designated initializers: a new C99 feature
 * that we can probably ignore.
 *
 * A designation can be used when initializing members of a struct
 * in some nice, yet confusing ways. Chances the students will never
 * use them, as <i>I've</i> only just read about them. The April
 * 2001 issue of <u>C++ Users Journal</u> has an article on them,
 * starting on page 56. Here is an example on page 58:
 *
 *     struct S1 {
 *         int i;
 *         float f;
 *         int a[2];
 *     };
 *
 *     struct S1 x = {
 *         .f=3.1,
 *         .i=2,
 *         .a[1]=9
 *     };
 *
 * Using this, you can initialize members by name. Interesting, but
 * not likely to be used.
 */
designation
    : designatorList ASSIGN
    ;

/** (6.7.8) designator-list: list of designators */
designatorList
    : (designator)+
    ;

/** (6.7.8) designator: used to initialize array members. */
designator
    : arrayDesignator
    | DOT IDENTIFIER
    ;

arrayDesignator
    : LBRACKET! (constantExpr)? RBRACKET!
    ;

/* A.2.3 Statements */

/** (6.8) statement: allows interesting things to happen. */
statement
// : labeledStatement
: compoundStatement
| expressionStatement
| selectionStatement
| iterationStatement
| jumpStatement
;

/* (6.8.1) labeled-statement: needed for <b>switch</b> blocks.
labeledStatement
: CASE^ constantExpr COLON! statement
| DEFAULT^ COLON! statement
;
*/

/** (6.8.2) compound-statement: a block of statements. */
compoundStatement
: LBRACE! blockItemList RBRACE! (SL_COMMENT!)?
{ #compoundStatement = #([BLOCK_STATEMENT, "BLOCK_STATEMENT"], #compoundStatement); }
;

/** (6.8.2) block-item-list: list of statements in a block. */
blockItemList
: (blockItem)*
;

```

```

/**
 * (6.8.2) block-item: declarations can be interspersed with statements now,
 * as in C++.
 *
 * Note: This must be kept up in sync with 'declaration'.
 */
blockItem
: (USING) => declaration
 | (TYPEDEF) => declaration
 | (declarationSpecifier) => declaration
 | statement
;

/**
 * (6.8.3) expression-statement: any expression can be a statement.
 *
 * Consider: the expression '2+2' is also the valid statement '2+2;'
 */
expressionStatement
: SEMI!
 | expression SEMI!
   { #expressionStatement = #[[EXPRESSION_STATEMENT, "EXPRESSION_STATEMENT"], #expressionStatement); }
;

/** (6.8.4) selection-statement: switch's and if's. */
selectionStatement
: ifNode:IF~ LPAREN! expression RPAREN! statement
  { if (!restriction.isEnabled(Restriction.IF))
    reportError ("'if' statements", ErrorCodeTable.ARE_DISABLED_2_PARAMS, ifNode.getLine(),
                ifNode.getColumn()); }
  ( options {
    warnWhenFollowAmbig = false;
  }
  : elseNode:ELSE! statement
  { if (!restriction.isEnabled(Restriction.IF_ELSE))
    reportError ("'if-then-else' statements", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
                elseNode.getLine(), elseNode.getColumn()); }
  )?

 | s:SWITCH~ LPAREN! expression RPAREN! LBRACE! (casesGroup)* RBRACE!
  { if (!restriction.isEnabled(Restriction.SWITCH))
    reportError ("'switch' statements", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
                s.getLine(), s.getColumn()); }
;

/** This was lifted by Peter from the ANTLR example for parsing the Java programming language. */
casesGroup
: ( // We're overriding the conflict here, see java example!
  options {
    warnWhenFollowAmbig = false;
  }
  :
  + case
  )+ caseSList
  {#casesGroup = #[[CASE_GROUP, "CASE_GROUP"], #casesGroup);}
;

acase
: (CASE~ expression | DEFAULT) COLON!
;

caseSList
: (statement)*
  {#caseSList = #[[SLIST, "SLIST"], #caseSList);}
;

/** (6.8.5) iteration-statement: loops of all kinds. */
iterationStatement
: w:WHILE~ LPAREN! expression RPAREN! statement
  { if (!restriction.isEnabled(Restriction.WHILE))
    reportError ("'while' loops", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
                w.getLine(), w.getColumn()); }
 | d:DO~ statement WHILE! LPAREN! expression RPAREN! SEMI!
  { if (!restriction.isEnabled(Restriction.DOWHILE))

```

```

        reportError ("do-while' loops", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
            d.getLine(), d.getColumn()); }
| (FOR LPAREN declarationSpecifier IDENTIFIER) =>
  f2:FOR~ LPAREN! declaration forCond SEMI! forIter RPAREN! statement
  { if (!restriction.isEnabled(Restriction.FOR_W_DEC))
    reportError ("for' loops with declarations", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
        f2.getLine(), f2.getColumn()); }
| f1:FOR~ LPAREN! forInit SEMI! forCond SEMI! forIter RPAREN! statement
  { if (!restriction.isEnabled(Restriction.FOR))
    reportError ("for' loops", ErrorCodeTable.ARE_DISABLED_2_PARAMS, f1.getLine(),
        f1.getColumn()); }
;

forInit
: (expression)?
  {#forInit = #[FOR_INIT,"FOR_INIT"], #forInit);}
;

forCond
: expression
  {#forCond = #[FOR_COND,"FOR_COND"], #forCond);}
;

forIter
: (expression)?
  {#forIter = #[FOR_ITER,"FOR_ITER"], #forIter);}
;

/** (6.8.6) jump-statement: changes where the next statement will occur. */
jumpStatement
// stop current loop and cause execution to begin at the top of the
// loop
: cont:CONTINUE~ SEMI!
  { if (!restriction.isEnabled(Restriction.CONTINUE))
    reportError ("Continue statements", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
        cont.getLine(), cont.getColumn()); }

// stop current loop and case execution to begin at the end of the
// current loop.
| brk:BREAK~ SEMI!
  { if (!restriction.isEnabled(Restriction.BREAK))
    reportError ("Break statements", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
        brk.getLine(), brk.getColumn()); }

// cause execution to stop in the current function. Execution will
// continue in the calling function.
| ret:RETURN~ (expression)? SEMI!
  { if (!restriction.isEnabled(Restriction.RETURN))
    reportError ("Return statements", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
        ret.getLine(), ret.getColumn()); }
;

/* A.2.4 External definitions */

/** (6.9) translation-unit: a single source file. */
translationUnit
: (preprocessingFile)* (externalDeclaration)*
;

/**
 * (6.9) external-declaration: declarations for types that are not defined
 * (storage-allocated) in this file.
 */
externalDeclaration
: ((declarationSpecifier)+ IDENTIFIER LPAREN) => functionDefinition
| declaration

// For globally-scoped null statements
| SEMI!
;

/**
 * (6.9.1) function-definition: the implementation of a function.

```

```

*
* The grammar appears to allow for K&R-style variable declarations,
* which I had thought were (finally) removed. We're not supporting
* K&R style, only ANSI-style.
*
* This isn't a direct equivalent of the standard, but it's sufficient.
*/
functionDefinition
: (declarationSpecifier)+ IDENTIFIER LPAREN! (parameterList)? RPAREN!
  (compoundStatement
  { #functionDefinition = #[[FUNCTION, "FUNCTION"], #functionDefinition]; }
  | SEMI!
  { #functionDefinition = #[[FUNCTION_PROTOTYPE, "FUNCTION_PROTOTYPE"], #functionDefinition]; }
  )
;

//
// A.3 Preprocessing Directives
//
// Since we're handling the preprocessor stuff at the same time as the normal
// lexer, we can't directly copy the standard. For example,
// 'preprocessing-file' says that 'group' should be optional. If we try
// that, however, we get ambiguity errors from ANTLR. Thus, some of the
// "optional" items are made required, so that we can tell whether or not
// they're present.
//

/** (6.10) preprocessing-file: a single source file. */
preprocessingFile
: group
;

/** (6.10) group: a group of preprocessor statements. */
group
: groupPart
;

/**
* (6.10) group-part: #if blocks, control lines, normal text, etc.
*
* We only care about '#include' statements, so we're doing
* a small subset.
*/
groupPart
: controlLine
;

/**
* (6.10) control-line: "commands" for the preprocessor, which include
* '#include' statements, '#define' statements, macros, and other
* preprocessor commands not dealing with '#if'.
*/
controlLine
: POUND_INCLUDE_DIRECTIVE
  { #controlLine = #[[INCLUDE, "INCLUDE"], #controlLine]; }
;

```

J.4 ANTLR-produced Output File for the *CS1 Sandbox*'s Parser (CParser.java)

```
// $ANTLR 2.7.1: "CParser.g" -> "CParser.java"$

package sandbox.compiler.frontend;

import java.util.*;
import sandbox.compiler.*;
import sandbox.client.ErrorCodeTable;

import antlr.TokenBuffer;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;
import antlr.ANTLRException;
import antlr.LLkParser;
import antlr.Token;
import antlr.TokenStream;
import antlr.RecognitionException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.ParserSharedInputState;
import antlr.collections.impl.BitSet;
import antlr.collections.AST;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

/**
 * Contains the grammar for the C Interpreter. The grammar (or a close
 * facsimile) is specified in docs/internal/grammar.html.
 */
public class CParser extends antlr.LLkParser
    implements CParserTokenTypes
{
    /** The errors that are observed during the parse phase. */
    private Vector errorVector = new Vector();

    /** The C language features that are allowed/forbidden. */
    private Restriction restriction;

    /** Are we currently parsing a Typedef? */
    boolean inTypedef = false;

    /** The CLexer that we're using to get tokens from/add types to. */
    private CLexer lexer;

    /**
     * Returns a vector of each error seen during compilation.
     */
    public Vector getErrorVector() {
        return(errorVector);
    }

    /**
     * Sets a reference to the current restriction object
     */
    public void setRestriction(Restriction restr) {
        restriction = restr;
    }

    /**
     * ReportError method, supporting the error reporting similiar to what ANTLR does.
     * We added support for the line number and column via this method.
     */
    public void reportError (java.lang.String errMsg, int code, int line, int col) {
        AntlrError error = new AntlrError ();
        error.setType("level");
    }
}

```

```

        error.setErrorMessage(errMsg);
        error.setErrorCode(code);
        error.setLineNumber(line);
        error.setColumnNumber(col);
        errorVector.add(error);
    }

    /**
     * Overridden reportError method, supporting the error reporting via ANTLR
     */
    public void reportError (int code, int line, int col) {
        AntlrError error = new AntlrError ();
        error.setType("level");
        error.setErrorCode (code);
        error.setLineNumber(line);
        error.setColumnNumber(col);
        errorVector.add(error);
    }

    /**
     * Overridden reportError method, supporting the error reporting via ANTLR
     */
    public void reportError(RecognitionException ex) {
        String errorMsg;
        AntlrError error = new AntlrError();
        error.setType("recognition exception - parser error");

        // tack "-- parser" onto the raw message so that it can be classified later
        errorMsg = new String(ex.getMessage() + "-- parser");
        error.setErrorMessage(errorMsg);
        error.setLineNumber(ex.getLine());
        error.setColumnNumber(ex.getColumn());
        errorVector.add(error);
    }

    public void reportError(String s) {
        AntlrError error = new AntlrError();
        error.setType("error - parser error");
        error.setErrorMessage(s);
        errorVector.add(error);
    }

    public void reportWarning(String s) {
        AntlrError error = new AntlrError();
        error.setType("warning");
        error.setErrorMessage(s);
        errorVector.add(error);
    }

    /**
     * Set the lexer to use when adding types (necessary for typedef support).
     */
    public void setLexer(CLexer l) {
        lexer = l;
    }

    /**
     * Add the type "type" to the lexer so that we can properly use it in
     * future variable declarations.
     */
    private void addType (String type) {
        // grab the lexer so we can add the type to the lexer.
        //lexer.getLibraryManager().getTypeManager().registerTypeName(type);
    }
}

protected CParser(TokenBuffer tokenBuf, int k) {
    super(tokenBuf,k);
    tokenNames = _tokenNames;
}

```

```

}

public CParser(TokenBuffer tokenBuf) {
    this(tokenBuf,3);
}

protected CParser(TokenStream lexer, int k) {
    super(lexer,k);
    tokenNames = _tokenNames;
}

public CParser(TokenStream lexer) {
    this(lexer,3);
}

public CParser(ParserSharedInputState state) {
    super(state,3);
    tokenNames = _tokenNames;
}

public final void program() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST program_AST = null;

    try {        // for error handling
        translationUnit();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        SandboxAST tmp1_AST = null;
        tmp1_AST = (SandboxAST)astFactory.create(LT(1));
        match(Token.EOF_TYPE);
        if ( inputState.guessing==0 ) {
            program_AST = (SandboxAST)currentAST.root;
            program_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
                SandboxAST)astFactory.create(PROGRAM,"PROGRAM")).add(program_AST));
            currentAST.root = program_AST;
            currentAST.child = program_AST!=null &&program_AST.getFirstChild()!=null ?
                program_AST.getFirstChild() : program_AST;
            currentAST.advanceChildToEnd();
        }
        program_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_0);
        } else {
            throw ex;
        }
    }
    returnAST = program_AST;
}

/** (6.9) translation-unit: a single source file. */
public final void translationUnit() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST translationUnit_AST = null;

    try {        // for error handling
        {
        _loop190:
        do {
            if ((LA(1)==POUND_INCLUDE_DIRECTIVE)) {
                preprocessingFile();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
            }
        }
        }
    }
}

```

```

        else {
            break _loop190;
        }
    } while (true);
}
{
_loop192:
do {
    if ((_tokenSet_1.member(LA(1)))) {
        externalDeclaration();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
    }
    else {
        break _loop192;
    }
} while (true);
}
translationUnit_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_0);
    } else {
        throw ex;
    }
}
returnAST = translationUnit_AST;
}

/** A.2.1 Expressions */
public final void primaryExpr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST primaryExpr_AST = null;

    try { // for error handling
        switch ( LA(1) ) {
        case IDENTIFIER:
            {
                SandboxAST tmp2_AST = null;
                if (inputState.guessing==0) {
                    tmp2_AST = (SandboxAST)astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp2_AST);
                }
                match(IDENTIFIER);
                primaryExpr_AST = (SandboxAST)currentAST.root;
                break;
            }
        case FLOATING_CONSTANT:
        case FALSE:
        case TRUE:
        case INTEGER_CONSTANT:
        case CHAR_CONSTANT:
            {
                constant();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
                primaryExpr_AST = (SandboxAST)currentAST.root;
                break;
            }
        case STRING_LITERAL:
            {
                SandboxAST tmp3_AST = null;
                if (inputState.guessing==0) {
                    tmp3_AST = (SandboxAST)astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp3_AST);
                }
            }
        }
    }
}

```

```

    }
    match(StringLiteral);
    primaryExpr_AST = (SandboxAST)currentAST.root;
    break;
}
case LPAREN:
{
    SandboxAST tmp4_AST = null;
    tmp4_AST = (SandboxAST)astFactory.create(LT(1));
    match(LPAREN);
    expression();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp5_AST = null;
    tmp5_AST = (SandboxAST)astFactory.create(LT(1));
    match(RPAREN);
    primaryExpr_AST = (SandboxAST)currentAST.root;
    break;
}
default:
{
    throw new NoViableAltException(LT(1), getFilename());
}
}
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_2);
    } else {
        throw ex;
    }
}
returnAST = primaryExpr_AST;
}

/** (6.4.4) constant: */
public final void constant() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST constant_AST = null;

    try { // for error handling
        switch ( LA(1) ) {
        case INTEGER_CONSTANT:
        {
            SandboxAST tmp6_AST = null;
            if (inputState.guessing==0) {
                tmp6_AST = (SandboxAST)astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp6_AST);
            }
            match(INTEGER_CONSTANT);
            constant_AST = (SandboxAST)currentAST.root;
            break;
        }
        case FLOATING_CONSTANT:
        {
            SandboxAST tmp7_AST = null;
            if (inputState.guessing==0) {
                tmp7_AST = (SandboxAST)astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp7_AST);
            }
            match(FLOATING_CONSTANT);
            constant_AST = (SandboxAST)currentAST.root;
            break;
        }
        case CHAR_CONSTANT:
        {
            SandboxAST tmp8_AST = null;
            if (inputState.guessing==0) {
                tmp8_AST = (SandboxAST)astFactory.create(LT(1));

```

```

        astFactory.addASTChild(currentAST, tmp8_AST);
    }
    match(CHAR_CONSTANT);
    constant_AST = (SandboxAST)currentAST.root;
    break;
}
case FALSE:
case TRUE:
{
    {
        switch ( LA(1) ) {
        case TRUE:
        {
            SandboxAST tmp9_AST = null;
            if (inputState.guessing==0) {
                tmp9_AST = (SandboxAST)astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp9_AST);
            }
            match(TRUE);
            break;
        }
        case FALSE:
        {
            SandboxAST tmp10_AST = null;
            if (inputState.guessing==0) {
                tmp10_AST = (SandboxAST)astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp10_AST);
            }
            match(FALSE);
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
        }
        constant_AST = (SandboxAST)currentAST.root;
        break;
    }
    default:
    {
        throw new NoViableAltException(LT(1), getFilename());
    }
}
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_2);
    }
    else {
        throw ex;
    }
}
returnAST = constant_AST;
}
}

/** (6.5.17) expression: */
public final void expression() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST expression_AST = null;

    try { // for error handling
        assignmentExpr();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        {
        _loop71:
        do {
            if ((LA(1)==COMMA)) {

```

```

        SandboxAST tmp11_AST = null;
        if (inputState.guessing==0) {
            tmp11_AST = (SandboxAST)astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp11_AST);
        }
        match(COMMA);
        assignmentExpr();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
    }
    else {
        break _loop71;
    }
}
} while (true);
}
expression_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_3);
    } else {
        throw ex;
    }
}
returnAST = expression_AST;
}
}

/**
 * (6.5.2) postfix-expr:
 *
 * Postfix Expressions include postfix ++ and --, structure (union, class,
 * etc.) member access, function calls, array accesses, and the like.
 *
 * Array accesses are handled in arrayExpr.
 *
 * Shamelessly copied from the example grammar "java.g".
 */
public final void postfixExpr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST postfixExpr_AST = null;
    SandboxAST t_AST = null;
    SandboxAST e_AST = null;
    SandboxAST t1_AST = null;
    SandboxAST e1_AST = null;
    SandboxAST t2_AST = null;
    SandboxAST e2_AST = null;
    SandboxAST t3_AST = null;
    SandboxAST e3_AST = null;
    SandboxAST i_AST = null;
    SandboxAST a_AST = null;
    Token in = null;
    SandboxAST in_AST = null;
    Token de = null;
    SandboxAST de_AST = null;

    try { // for error handling
        switch ( LA(1)) {
            case ENUM:
            case STRUCT:
            case TYPENAME:
            {
                typeSpecifier();
                if (inputState.guessing==0) {
                    t_AST = (SandboxAST)returnAST;
                    astFactory.addASTChild(currentAST, returnAST);
                }
                SandboxAST tmp12_AST = null;
                tmp12_AST = (SandboxAST)astFactory.create(LT(1));

```

```

match(LPAREN);
expression();
if (inputState.guessing==0) {
    e_AST = (SandboxAST)returnAST;
    astFactory.addASTChild(currentAST, returnAST);
}
SandboxAST tmp13_AST = null;
tmp13_AST = (SandboxAST)astFactory.create(LT(1));
match(RPAREN);
if ( inputState.guessing==0 ) {
    postfixExpr_AST = (SandboxAST)currentAST.root;
    postfixExpr_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
        SandboxAST)astFactory.create(CAST,"CAST")).add(postfixExpr_AST));
    currentAST.root = postfixExpr_AST;
    currentAST.child = postfixExpr_AST!=null &&postfixExpr_AST.getFirstChild()!=null ?
        postfixExpr_AST.getFirstChild() : postfixExpr_AST;
    currentAST.advanceChildToEnd();
}
postfixExpr_AST = (SandboxAST)currentAST.root;
break;
}
case STATIC_CAST:
{
    SandboxAST tmp14_AST = null;
    if (inputState.guessing==0) {
        tmp14_AST = (SandboxAST)astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp14_AST);
    }
    match(STATIC_CAST);
    SandboxAST tmp15_AST = null;
    if (inputState.guessing==0) {
        tmp15_AST = (SandboxAST)astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp15_AST);
    }
    match(LT);
    typeSpecifier();
    if (inputState.guessing==0) {
        t1_AST = (SandboxAST)returnAST;
        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp16_AST = null;
    if (inputState.guessing==0) {
        tmp16_AST = (SandboxAST)astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp16_AST);
    }
    match(GT);
    SandboxAST tmp17_AST = null;
    if (inputState.guessing==0) {
        tmp17_AST = (SandboxAST)astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp17_AST);
    }
    match(LPAREN);
    expression();
    if (inputState.guessing==0) {
        e1_AST = (SandboxAST)returnAST;
        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp18_AST = null;
    tmp18_AST = (SandboxAST)astFactory.create(LT(1));
    match(RPAREN);
    if ( inputState.guessing==0 ) {
        postfixExpr_AST = (SandboxAST)currentAST.root;
        postfixExpr_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
            SandboxAST)astFactory.create(CAST,"CAST")).add(postfixExpr_AST));
        currentAST.root = postfixExpr_AST;
        currentAST.child = postfixExpr_AST!=null &&postfixExpr_AST.getFirstChild()!=null ?
            postfixExpr_AST.getFirstChild() : postfixExpr_AST;
        currentAST.advanceChildToEnd();
    }
    postfixExpr_AST = (SandboxAST)currentAST.root;
    break;
}
case CONST_CAST:
{

```

```

SandboxAST tmp19_AST = null;
if (inputState.guessing==0) {
    tmp19_AST = (SandboxAST)astFactory.create(LT(1));
    astFactory.addASTChild(currentAST, tmp19_AST);
}
match(CONST_CAST);
SandboxAST tmp20_AST = null;
if (inputState.guessing==0) {
    tmp20_AST = (SandboxAST)astFactory.create(LT(1));
    astFactory.addASTChild(currentAST, tmp20_AST);
}
match(LT);
typeSpecifier();
if (inputState.guessing==0) {
    t2_AST = (SandboxAST)returnAST;
    astFactory.addASTChild(currentAST, returnAST);
}
SandboxAST tmp21_AST = null;
if (inputState.guessing==0) {
    tmp21_AST = (SandboxAST)astFactory.create(LT(1));
    astFactory.addASTChild(currentAST, tmp21_AST);
}
match(GT);
SandboxAST tmp22_AST = null;
if (inputState.guessing==0) {
    tmp22_AST = (SandboxAST)astFactory.create(LT(1));
    astFactory.addASTChild(currentAST, tmp22_AST);
}
match(LPAREN);
expression();
if (inputState.guessing==0) {
    e2_AST = (SandboxAST)returnAST;
    astFactory.addASTChild(currentAST, returnAST);
}
SandboxAST tmp23_AST = null;
tmp23_AST = (SandboxAST)astFactory.create(LT(1));
match(RPAREN);
if ( inputState.guessing==0 ) {
    postfixExpr_AST = (SandboxAST)currentAST.root;
    postfixExpr_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
        SandboxAST)astFactory.create(CAST,"CAST")).add(postfixExpr_AST));
    currentAST.root = postfixExpr_AST;
    currentAST.child = postfixExpr_AST!=null &&postfixExpr_AST.getFirstChild()!=null ?
        postfixExpr_AST.getFirstChild() : postfixExpr_AST;
    currentAST.advanceChildToEnd();
}
postfixExpr_AST = (SandboxAST)currentAST.root;
break;
}
case REINTERPRET_CAST:
{
    SandboxAST tmp24_AST = null;
    if (inputState.guessing==0) {
        tmp24_AST = (SandboxAST)astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp24_AST);
    }
    match(REINTERPRET_CAST);
    SandboxAST tmp25_AST = null;
    if (inputState.guessing==0) {
        tmp25_AST = (SandboxAST)astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp25_AST);
    }
    match(LT);
    typeSpecifier();
    if (inputState.guessing==0) {
        t3_AST = (SandboxAST)returnAST;
        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp26_AST = null;
    if (inputState.guessing==0) {
        tmp26_AST = (SandboxAST)astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp26_AST);
    }
    match(GT);
}

```

```

SandboxAST tmp27_AST = null;
if (inputState.guessing==0) {
    tmp27_AST = (SandboxAST)astFactory.create(LT(1));
    astFactory.addASTChild(currentAST, tmp27_AST);
}
match(LPAREN);
expression();
if (inputState.guessing==0) {
    e3_AST = (SandboxAST)returnAST;
    astFactory.addASTChild(currentAST, returnAST);
}
SandboxAST tmp28_AST = null;
tmp28_AST = (SandboxAST)astFactory.create(LT(1));
match(RPAREN);
if ( inputState.guessing==0 ) {
    postfixExpr_AST = (SandboxAST)currentAST.root;
    postfixExpr_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
        SandboxAST)astFactory.create(CAST,"CAST")).add(postfixExpr_AST));
    currentAST.root = postfixExpr_AST;
    currentAST.child = postfixExpr_AST!=null &&postfixExpr_AST.getFirstChild()!=null ?
        postfixExpr_AST.getFirstChild() : postfixExpr_AST;
    currentAST.advanceChildToEnd();
}
postfixExpr_AST = (SandboxAST)currentAST.root;
break;
}
case FLOATING_CONSTANT:
case FALSE:
case TRUE:
case LPAREN:
case IDENTIFIER:
case INTEGER_CONSTANT:
case CHAR_CONSTANT:
case STRING_LITERAL:
{
    primaryExpr();
    if (inputState.guessing==0) {
        i_AST = (SandboxAST)returnAST;
        astFactory.addASTChild(currentAST, returnAST);
    }
}
_loop8:
do {
    switch ( LA(1) ) {
        case DOT:
        {
            SandboxAST tmp29_AST = null;
            if (inputState.guessing==0) {
                tmp29_AST = (SandboxAST)astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp29_AST);
            }
            match(DOT);
            SandboxAST tmp30_AST = null;
            if (inputState.guessing==0) {
                tmp30_AST = (SandboxAST)astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp30_AST);
            }
            match(IDENTIFIER);
            break;
        }
        case COLON_COLON:
        {
            SandboxAST tmp31_AST = null;
            if (inputState.guessing==0) {
                tmp31_AST = (SandboxAST)astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp31_AST);
            }
            match(COLON_COLON);
            SandboxAST tmp32_AST = null;
            if (inputState.guessing==0) {
                tmp32_AST = (SandboxAST)astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp32_AST);
            }
            match(IDENTIFIER);
        }
    }
}

```

```

    break;
  }
  case LPAREN:
  {
    SandboxAST tmp33_AST = null;
    if (inputState.guessing==0) {
      tmp33_AST = (SandboxAST)astFactory.create(LT(1));
      astFactory.makeASTRoot(currentAST, tmp33_AST);
    }
    match(LPAREN);
    {
      switch ( LA(1)) {
      case FLOATING_CONSTANT:
      case CONST_CAST:
      case ENUM:
      case FALSE:
      case REINTERPRET_CAST:
      case SIZEOF:
      case STATIC_CAST:
      case STRUCT:
      case TYPENAME:
      case TRUE:
      case LPAREN:
      case INCR:
      case DECR:
      case PLUS:
      case MINUS:
      case TILDE:
      case BANG:
      case IDENTIFIER:
      case INTEGER_CONSTANT:
      case CHAR_CONSTANT:
      case STRING_LITERAL:
      {
        argumentExprList();
        if (inputState.guessing==0) {
          astFactory.addASTChild(currentAST, returnAST);
        }
        break;
      }
      case RPAREN:
      {
        break;
      }
      default:
      {
        throw new NoViableAltException(LT(1), getFilename());
      }
    }
    SandboxAST tmp34_AST = null;
    tmp34_AST = (SandboxAST)astFactory.create(LT(1));
    match(RPAREN);
    if ( inputState.guessing==0 ) {
      postfixExpr_AST = (SandboxAST)currentAST.root;
      postfixExpr_AST = (SandboxAST)astFactory.make( (new ASTArray(3)).add((
        SandboxAST)astFactory.create(CALL,"CALL").add(i_AST).add(a_AST));
      currentAST.root = postfixExpr_AST;
      currentAST.child = postfixExpr_AST!=null &&postfixExpr_AST.getFirstChild()!=null ?
        postfixExpr_AST.getFirstChild() : postfixExpr_AST;
      currentAST.advanceChildToEnd();
    }
    break;
  }
  default:
  {
    break _loop8;
  }
} while (true);
}
{
  switch ( LA(1)) {
  case INCR:

```



```

        {
            throw new NoViableAltException(LT(1), getFilename());
        }
    }
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_4);
    } else {
        throw ex;
    }
}
returnAST = postfixExpr_AST;
}

/**
 * (6.7.2) type-specifier: types that can be used in declarations.
 */
public final void typeSpecifier() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST typeSpecifier_AST = null;

    try { // for error handling
        switch ( LA(1) ) {
            case TYPENAME:
            {
                SandboxAST tmp35_AST = null;
                if (inputState.guessing==0) {
                    tmp35_AST = (SandboxAST)astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp35_AST);
                }
                match(TYPENAME);
                typeSpecifier_AST = (SandboxAST)currentAST.root;
                break;
            }
            case STRUCT:
            {
                structSpecifier();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
                typeSpecifier_AST = (SandboxAST)currentAST.root;
                break;
            }
            case ENUM:
            {
                enumSpecifier();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
                typeSpecifier_AST = (SandboxAST)currentAST.root;
                break;
            }
            default:
            {
                throw new NoViableAltException(LT(1), getFilename());
            }
        }
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_5);
        } else {
            throw ex;
        }
    }
    returnAST = typeSpecifier_AST;
}

```

```

    }

/** (6.5.2) argument-expr-list: list of the actual arguments to a function. */
public final void argumentExprList() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST argumentExprList_AST = null;

    try {        // for error handling
        assignmentExpr();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        {
        _loop28:
        do {
            if ((LA(1)==COMMA)) {
                SandboxAST tmp36_AST = null;
                tmp36_AST = (SandboxAST)astFactory.create(LT(1));
                match(COMMA);
                assignmentExpr();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
            }
            else {
                break _loop28;
            }
        } while (true);
        }
        if ( inputState.guessing==0 ) {
            argumentExprList_AST = (SandboxAST)currentAST.root;
            argumentExprList_AST = (SandboxAST)astFactory.make( ( new ASTArray(2) ).add((
                SandboxAST)astFactory.create(ARGUMENTS,"ARGUMENTS")).add(argumentExprList_AST));
            currentAST.root = argumentExprList_AST;
            currentAST.child = argumentExprList_AST!=null &&argumentExprList_AST.getFirstChild()!=null ?
                argumentExprList_AST.getFirstChild() : argumentExprList_AST;
            currentAST.advanceChildToEnd();
        }
        argumentExprList_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_6);
        } else {
            throw ex;
        }
    }
    returnAST = argumentExprList_AST;
}

public final void memberAccess() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST memberAccess_AST = null;

    try {        // for error handling
        boolean synPredMatched14 = false;
        if ((((_tokenSet_7.member(LA(1))) && (_tokenSet_8.member(LA(2))) && (_tokenSet_9.member(LA(3)))))) {
            int _m14 = mark();
            synPredMatched14 = true;
            inputState.guessing++;
            try {
                {
                primaryExpr();
                {
                int _cnt13=0;
                _loop13:
                do {

```

```

        if ((LA(1)==DOT)) {
            match(DOT);
            match(IDENTIFIER);
        }
        else {
            if ( _cnt13>=1 ) { break _loop13; } else {throw new NoViableAltException(LT(1), getFilename());}
        }

        _cnt13++;
    } while (true);
}
match(LPAREN);
}
}
catch (RecognitionException pe) {
    synPredMatched14 = false;
}
rewind(_m14);
inputState.guessing--;
}
if ( synPredMatched14 ) {
    callMemberFunction();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    memberAccess_AST = (SandboxAST)currentAST.root;
}
else if (( _tokenSet_7.member(LA(1)) && ( _tokenSet_8.member(LA(2)) ) && ( _tokenSet_9.member(LA(3)) )) {
    primaryExpr();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    {
    }
    int _cnt16=0;
    _loop16:
    do {
        if ((LA(1)==DOT)) {
            SandboxAST tmp37_AST = null;
            if (inputState.guessing==0) {
                tmp37_AST = (SandboxAST)astFactory.create(LT(1));
                astFactory.makeASTRoot(currentAST, tmp37_AST);
            }
            match(DOT);
            SandboxAST tmp38_AST = null;
            if (inputState.guessing==0) {
                tmp38_AST = (SandboxAST)astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp38_AST);
            }
            match(IDENTIFIER);
        }
        else {
            if ( _cnt16>=1 ) { break _loop16; } else {throw new NoViableAltException(LT(1), getFilename());}
        }

        _cnt16++;
    } while (true);
}
memberAccess_AST = (SandboxAST)currentAST.root;
}
else {
    throw new NoViableAltException(LT(1), getFilename());
}
}
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_0);
    }
    else {
        throw ex;
    }
}
}
returnAST = memberAccess_AST;

```



```

    }
    default:
    {
        throw new NoViableAltException(LT(1), getFilename());
    }
}

SandboxAST tmp42_AST = null;
tmp42_AST = (SandboxAST)astFactory.create(LT(1));
match(RPAREN);
if ( inputState.guessing==0 ) {
    callMemberFunction_AST = (SandboxAST)currentAST.root;
    callMemberFunction_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
        SandboxAST)astFactory.create(CALL,"CALL")).add(callMemberFunction_AST));
    currentAST.root = callMemberFunction_AST;
    currentAST.child = callMemberFunction_AST!=null &&callMemberFunction_AST.getFirstChild()!=null ?
        callMemberFunction_AST.getFirstChild() : callMemberFunction_AST;
    currentAST.advanceChildToEnd();
}
callMemberFunction_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_0);
    } else {
        throw ex;
    }
}
returnAST = callMemberFunction_AST;
}

/**
 * Handle array access.  If a '[' is observed, the next token must be
 * an expression of some form so that we know where to look in the array.
 */
public final void arrayExpr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST arrayExpr_AST = null;

    try { // for error handling
        boolean synPredMatched23 = false;
        if ((((_tokenSet_10.member(LA(1))) && (_tokenSet_11.member(LA(2))) && (_tokenSet_12.member(LA(3))))) {
            int _m23 = mark();
            synPredMatched23 = true;
            inputState.guessing++;
            try {
                {
                    postfixExpr();
                    match(LBRACKET);
                }
            }
            catch (RecognitionException pe) {
                synPredMatched23 = false;
            }
            rewind(_m23);
            inputState.guessing--;
        }
        if ( synPredMatched23 ) {
            postfixExpr();
            if (inputState.guessing==0) {
                astFactory.addASTChild(currentAST, returnAST);
            }
            {
                _loop25:
                do {
                    if ((LA(1)==LBRACKET)) {
                        SandboxAST tmp43_AST = null;
                        tmp43_AST = (SandboxAST)astFactory.create(LT(1));
                        match(LBRACKET);
                        expression();
                    }
                }
            }
        }
    }
}

```

```

        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        SandboxAST tmp44_AST = null;
        tmp44_AST = (SandboxAST)astFactory.create(LT(1));
        match(RBRACKET);
    }
    else {
        break _loop25;
    }
} while (true);
}
if ( inputState.guessing==0 ) {
    arrayExpr_AST = (SandboxAST)currentAST.root;
    arrayExpr_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
        SandboxAST)astFactory.create(SUBSCRIPT,"SUBSCRIPT")).add(arrayExpr_AST));
    currentAST.root = arrayExpr_AST;
    currentAST.child = arrayExpr_AST!=null &&arrayExpr_AST.getFirstChild()!=null ?
        arrayExpr_AST.getFirstChild() : arrayExpr_AST;
    currentAST.advanceChildToEnd();
}
arrayExpr_AST = (SandboxAST)currentAST.root;
}
else if ((_tokenSet_10.member(LA(1))) && (_tokenSet_13.member(LA(2))) && (_tokenSet_12.member(LA(3)))) {
    postfixExpr();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    arrayExpr_AST = (SandboxAST)currentAST.root;
}
else {
    throw new NoViableAltException(LT(1), getFilename());
}
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_14);
    } else {
        throw ex;
    }
}
returnAST = arrayExpr_AST;
}

/** (6.5.16) assignment-expr: */
public final void assignmentExpr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST assignmentExpr_AST = null;
    Token assign = null;
    SandboxAST assign_AST = null;
    Token star = null;
    SandboxAST star_AST = null;
    Token slash = null;
    SandboxAST slash_AST = null;
    Token mod = null;
    SandboxAST mod_AST = null;
    Token add = null;
    SandboxAST add_AST = null;
    Token sub = null;
    SandboxAST sub_AST = null;

    try { // for error handling
        boolean synPredMatched67 = false;
        if (((_tokenSet_15.member(LA(1))) && (_tokenSet_16.member(LA(2))) && (_tokenSet_17.member(LA(3)))) {
            int _m67 = mark();
            synPredMatched67 = true;
            inputState.guessing++;
            try {

```

```

    {
    unaryExpr();
    {
    switch ( LA(1)) {
    case ASSIGN:
    {
    match(ASSIGN);
    break;
    }
    case STAR_ASSIGN:
    {
    match(STAR_ASSIGN);
    break;
    }
    case SLASH_ASSIGN:
    {
    match(SLASH_ASSIGN);
    break;
    }
    case PERCENT_ASSIGN:
    {
    match(PERCENT_ASSIGN);
    break;
    }
    case PLUS_ASSIGN:
    {
    match(PLUS_ASSIGN);
    break;
    }
    case MINUS_ASSIGN:
    {
    match(MINUS_ASSIGN);
    break;
    }
    default:
    {
    throw new NoViableAltException(LT(1), getFilename());
    }
    }
    }
}
catch (RecognitionException pe) {
    synPredMatched67 = false;
}
rewind(_m67);
inputState.guessing--;
}
if ( synPredMatched67 ) {
    unaryExpr();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    {
    switch ( LA(1)) {
    case ASSIGN:
    {
    assign = LT(1);
    if (inputState.guessing==0) {
        assign_AST = (SandboxAST)astFactory.create(assign);
        astFactory.makeASTRoot(currentAST, assign_AST);
    }
    match(ASSIGN);
    break;
    }
    case STAR_ASSIGN:
    {
    star = LT(1);
    if (inputState.guessing==0) {
        star_AST = (SandboxAST)astFactory.create(star);
        astFactory.makeASTRoot(currentAST, star_AST);
    }
    match(STAR_ASSIGN);
    break;
    }
    }
}

```

```

}
case SLASH_ASSIGN:
{
    slash = LT(1);
    if (inputState.guessing==0) {
        slash_AST = (SandboxAST)astFactory.create(slash);
        astFactory.makeASTRoot(currentAST, slash_AST);
    }
    match(SLASH_ASSIGN);
    break;
}
case PERCENT_ASSIGN:
{
    mod = LT(1);
    if (inputState.guessing==0) {
        mod_AST = (SandboxAST)astFactory.create(mod);
        astFactory.makeASTRoot(currentAST, mod_AST);
    }
    match(PERCENT_ASSIGN);
    break;
}
case PLUS_ASSIGN:
{
    add = LT(1);
    if (inputState.guessing==0) {
        add_AST = (SandboxAST)astFactory.create(add);
        astFactory.makeASTRoot(currentAST, add_AST);
    }
    match(PLUS_ASSIGN);
    break;
}
case MINUS_ASSIGN:
{
    sub = LT(1);
    if (inputState.guessing==0) {
        sub_AST = (SandboxAST)astFactory.create(sub);
        astFactory.makeASTRoot(currentAST, sub_AST);
    }
    match(MINUS_ASSIGN);
    break;
}
default:
{
    throw new NoViableAltException(LT(1), getFilename());
}
}
assignmentExpr();
if (inputState.guessing==0) {
    astFactory.addASTChild(currentAST, returnAST);
}
if ( inputState.guessing==0 ) {

    if (!restriction.isEnabled(Restriction.ASSIGNMENT) && assign != null)
        reportError ("Assignment statements (=)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
            assign.getLine(), assign.getColumn());

    if (!restriction.isEnabled(Restriction.MULT_ASSIGN) && star != null)
        reportError ("Multiplication/assignment statements (*=)",
            ErrorCodeTable.ARE_DISABLED_3_PARAMS, star.getLine(), star.getColumn());

    if (!restriction.isEnabled(Restriction.DIV_ASSIGN) && slash != null)
        reportError ("Division/assignment statements (/=)",
            ErrorCodeTable.ARE_DISABLED_3_PARAMS, slash.getLine(), slash.getColumn());

    if (!restriction.isEnabled(Restriction.MOD_ASSIGN) && mod != null)
        reportError ("Modulus/assignment statements (%=)",
            ErrorCodeTable.ARE_DISABLED_3_PARAMS, mod.getLine(), mod.getColumn());

    if (!restriction.isEnabled(Restriction.ADD_ASSIGN) && add != null)
        reportError ("Addition/assignment statements (+=)",
            ErrorCodeTable.ARE_DISABLED_3_PARAMS, add.getLine(), add.getColumn());

    if (!restriction.isEnabled(Restriction.SUB_ASSIGN) && sub != null)

```

```

        reportError ("Subtraction/assignment statements (-=)",
            ErrorCodeTable.ARE_DISABLED_3_PARAMS, sub.getLine(), sub.getColumn());
    }
    assignmentExpr_AST = (SandboxAST)currentAST.root;
}
else if ((_tokenSet_18.member(LA(1))) && (_tokenSet_19.member(LA(2))) && (_tokenSet_12.member(LA(3)))) {
    logicalOrExpr();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    assignmentExpr_AST = (SandboxAST)currentAST.root;
}
else {
    throw new NoViableAltException(LT(1), getFilename());
}
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_20);
    } else {
        throw ex;
    }
}
returnAST = assignmentExpr_AST;
}
}

/** (6.5.3) unary-expr: */
public final void unaryExpr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST unaryExpr_AST = null;
    Token preInc = null;
    SandboxAST preInc_AST = null;
    Token preDec = null;
    SandboxAST preDec_AST = null;

    try { // for error handling
        switch ( LA(1)) {
        case FLOATING_CONSTANT:
        case CONST_CAST:
        case ENUM:
        case FALSE:
        case REINTERPRET_CAST:
        case STATIC_CAST:
        case STRUCT:
        case TYPENAME:
        case TRUE:
        case LPAREN:
        case INCR:
        case DECR:
        case IDENTIFIER:
        case INTEGER_CONSTANT:
        case CHAR_CONSTANT:
        case STRING_LITERAL:
        {
            {
                switch ( LA(1)) {
                case INCR:
                {
                    preInc = LT(1);
                    if (inputState.guessing==0) {
                        preInc_AST = (SandboxAST)astFactory.create(preInc);
                        astFactory.makeASTRoot(currentAST, preInc_AST);
                    }
                }
                match(INCR);
                break;
            }
        }
        case DECR:
        {

```

```

preDec = LT(1);
if (inputState.guessing==0) {
    preDec_AST = (SandboxAST)astFactory.create(preDec);
    astFactory.makeASTRoot(currentAST, preDec_AST);
}
match(DECR);
break;
}
case FLOATING_CONSTANT:
case CONST_CAST:
case ENUM:
case FALSE:
case REINTERPRET_CAST:
case STATIC_CAST:
case STRUCT:
case TYPENAME:
case TRUE:
case LPAREN:
case IDENTIFIER:
case INTEGER_CONSTANT:
case CHAR_CONSTANT:
case STRING_LITERAL:
{
    break;
}
default:
{
    throw new NoViableAltException(LT(1), getFilename());
}
}
}
arrayExpr();
if (inputState.guessing==0) {
    astFactory.addASTChild(currentAST, returnAST);
}
if ( inputState.guessing==0 ) {

    if (!restriction.isEnabled(Restriction.PREFIX))
        if (preInc != null)
            reportError ("Prefix expressions", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
                preInc.getLine(), preInc.getColumn());
        else if (preDec != null)
            reportError ("Prefix expressions", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
                preDec.getLine(), preDec.getColumn());

}
unaryExpr_AST = (SandboxAST)currentAST.root;
break;
}
case PLUS:
case MINUS:
case TILDE:
case BANG:
{
    switch ( LA(1) ) {
    case PLUS:
    {
        SandboxAST tmp45_AST = null;
        if (inputState.guessing==0) {
            tmp45_AST = (SandboxAST)astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp45_AST);
        }
        match(PLUS);
        break;
    }
    case MINUS:
    {
        SandboxAST tmp46_AST = null;
        if (inputState.guessing==0) {
            tmp46_AST = (SandboxAST)astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp46_AST);
        }
        match(MINUS);
    }
    }
}

```

```

        break;
    }
    case TILDE:
    {
        SandboxAST tmp47_AST = null;
        if (inputState.guessing==0) {
            tmp47_AST = (SandboxAST)astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp47_AST);
        }
        match(TILDE);
        break;
    }
    case BANG:
    {
        SandboxAST tmp48_AST = null;
        if (inputState.guessing==0) {
            tmp48_AST = (SandboxAST)astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp48_AST);
        }
        match(BANG);
        break;
    }
    default:
    {
        throw new NoViableAltException(LT(1), getFilename());
    }
    }
    castExpr();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    unaryExpr_AST = (SandboxAST)currentAST.root;
    break;
}
default:
{
    throw new NoViableAltException(LT(1), getFilename());
}
}
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_14);
    } else {
        throw ex;
    }
}
returnAST = unaryExpr_AST;
}
}

/** (6.5.4) cast-expr: */
public final void castExpr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST castExpr_AST = null;
    Token typecast2 = null;
    SandboxAST typecast2_AST = null;
    Token typecast = null;
    SandboxAST typecast_AST = null;

    try { // for error handling
        boolean synPredMatched35 = false;
        if (((LA(1)==LPAREN) && (LA(2)==LPAREN) && (LA(3)==ENUM||LA(3)==STRUCT||LA(3)==TYPENAME))) {
            int _m35 = mark();
            synPredMatched35 = true;
            inputState.guessing++;
            try {
                {
                    match(LPAREN);
                    match(LPAREN);
                }
            }
        }
    }
}

```

```

        typeSpecifier();
    }
}
catch (RecognitionException pe) {
    synPredMatched35 = false;
}
rewind(_m35);
inputState.guessing--;
}
if ( synPredMatched35 ) {
    typecast2 = LT(1);
    if (inputState.guessing==0) {
        typecast2_AST = (SandboxAST)astFactory.create(typecast2);
    }
    match(LPAREN);
    SandboxAST tmp49_AST = null;
    tmp49_AST = (SandboxAST)astFactory.create(LT(1));
    match(LPAREN);
    typeSpecifier();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp50_AST = null;
    tmp50_AST = (SandboxAST)astFactory.create(LT(1));
    match(RPAREN);
    unaryExpr();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp51_AST = null;
    tmp51_AST = (SandboxAST)astFactory.create(LT(1));
    match(RPAREN);
    if ( inputState.guessing==0 ) {
        castExpr_AST = (SandboxAST)currentAST.root;

        castExpr_AST = (SandboxAST)astFactory.make( ( new ASTArray(2)).add((
            SandboxAST)astFactory.create(CAST,"CAST").add(castExpr_AST));
        if (!restriction.isEnabled(Restriction.TYPECAST) && typecast2 != null)
            reportError ("Explicit typecast statements", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
                typecast2.getLine(), typecast.getColumn());

        currentAST.root = castExpr_AST;
        currentAST.child = castExpr_AST!=null &&castExpr_AST.getFirstChild()!=null ?
            castExpr_AST.getFirstChild() : castExpr_AST;
        currentAST.advanceChildToEnd();
    }
    castExpr_AST = (SandboxAST)currentAST.root;
}
else {
    boolean synPredMatched37 = false;
    if (((LA(1)==LPAREN) && (LA(2)==ENUM||LA(2)==STRUCT||LA(2)==TYPENAME) && (_tokenSet_21.member(LA(3)))) {
        int _m37 = mark();
        synPredMatched37 = true;
        inputState.guessing++;
        try {
            {
                match(LPAREN);
                typeSpecifier();
            }
        }
        catch (RecognitionException pe) {
            synPredMatched37 = false;
        }
        rewind(_m37);
        inputState.guessing--;
    }
}
if ( synPredMatched37 ) {
    typecast = LT(1);
    if (inputState.guessing==0) {
        typecast_AST = (SandboxAST)astFactory.create(typecast);
    }
    match(LPAREN);
    typeSpecifier();
    if (inputState.guessing==0) {

```

```

        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp52_AST = null;
    tmp52_AST = (SandboxAST)astFactory.create(LT(1));
    match(RPAREN);
    unaryExpr();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    if ( inputState.guessing==0 ) {
        castExpr_AST = (SandboxAST)currentAST.root;

        castExpr_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
            SandboxAST)astFactory.create(CAST,"CAST")).add(castExpr_AST));
        if (!restriction.isEnabled(Restriction.TYPESCAST) && typecast != null)
            reportError ("Explicit typecast statements", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
                typecast.getLine(), typecast.getColumn());

        currentAST.root = castExpr_AST;
        currentAST.child = castExpr_AST!=null &&castExpr_AST.getFirstChild()!=null ?
            castExpr_AST.getFirstChild() : castExpr_AST;
        currentAST.advanceChildToEnd();
    }
    castExpr_AST = (SandboxAST)currentAST.root;
}
else if ((_tokenSet_15.member(LA(1))) && (_tokenSet_11.member(LA(2))) && (_tokenSet_12.member(LA(3)))) {
    unaryExpr();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    castExpr_AST = (SandboxAST)currentAST.root;
}
else if ((LA(1)==SIZEOF)) {
    sizeofExpr();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    castExpr_AST = (SandboxAST)currentAST.root;
}
else {
    throw new NoViableAltException(LT(1), getFilename());
}
}
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_14);
    } else {
        throw ex;
    }
}
returnAST = castExpr_AST;
}
}

/** Handle <b>sizeof</b> expressions. This is an internal function. */
public final void sizeofExpr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST sizeofExpr_AST = null;

    try { // for error handling
        SandboxAST tmp53_AST = null;
        if (inputState.guessing==0) {
            tmp53_AST = (SandboxAST)astFactory.create(LT(1));
            astFactory.makeASTRoot(currentAST, tmp53_AST);
        }
        match(SIZEOF);
        unaryExpr();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
    }
}

```

```

        sizeofExpr_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_14);
        } else {
            throw ex;
        }
    }
    returnAST = sizeofExpr_AST;
}

/** (6.5.5) multiplicative-expr: */
public final void multiplicativeExpr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST multiplicativeExpr_AST = null;
    Token mult = null;
    SandboxAST mult_AST = null;
    Token div = null;
    SandboxAST div_AST = null;
    Token mod = null;
    SandboxAST mod_AST = null;

    try { // for error handling
        castExpr();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        {
        _loop41:
        do {
            if ((LA(1)==STAR||LA(1)==SLASH||LA(1)==PERCENT)) {
                {
                switch ( LA(1) ) {
                case STAR:
                {
                    mult = LT(1);
                    if (inputState.guessing==0) {
                        mult_AST = (SandboxAST)astFactory.create(mult);
                        astFactory.makeASTRoot(currentAST, mult_AST);
                    }
                    match(STAR);
                    break;
                }
                case SLASH:
                {
                    div = LT(1);
                    if (inputState.guessing==0) {
                        div_AST = (SandboxAST)astFactory.create(div);
                        astFactory.makeASTRoot(currentAST, div_AST);
                    }
                    match(SLASH);
                    break;
                }
                case PERCENT:
                {
                    mod = LT(1);
                    if (inputState.guessing==0) {
                        mod_AST = (SandboxAST)astFactory.create(mod);
                        astFactory.makeASTRoot(currentAST, mod_AST);
                    }
                    match(PERCENT);
                    break;
                }
                default:
                {
                    throw new NoViableAltException(LT(1), getFilename());
                }
                }
            }
        }
    }
}

```

```

        castExpr();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
    }
    else {
        break _loop41;
    }
} while (true);
}
if ( inputState.guessing==0 ) {

    if (!restriction.isEnabled(Restriction.MULTIPLICATION) && mult != null)
        reportError ("Multiplication statements (*)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
            mult.getLine(), mult.getColumn());

    if (!restriction.isEnabled(Restriction.DIVISION) && div != null)
        reportError ("Division statements (/)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
            div.getLine(), div.getColumn());

    if (!restriction.isEnabled(Restriction.MODULUS) && mod != null)
        reportError ("Modulus statements (%)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
            mod.getLine(), mod.getColumn());

}
multiplicativeExpr_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_22);
    } else {
        throw ex;
    }
}
}
returnAST = multiplicativeExpr_AST;
}

/** (6.5.6) additive-expr: */
public final void additiveExpr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST additiveExpr_AST = null;
    Token addPlus = null;
    SandboxAST addPlus_AST = null;
    Token addMinus = null;
    SandboxAST addMinus_AST = null;

    try { // for error handling
        multiplicativeExpr();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
    }
    {
        _loop45:
        do {
            if ((LA(1)==PLUS||LA(1)==MINUS)) {
                {
                    switch ( LA(1) ) {
                    case PLUS:
                    {
                        addPlus = LT(1);
                        if (inputState.guessing==0) {
                            addPlus_AST = (SandboxAST)astFactory.create(addPlus);
                            astFactory.makeASTRoot(currentAST, addPlus_AST);
                        }
                    }
                    match(PLUS);
                    break;
                }
                case MINUS:
                {

```

```

        addMinus = LT(1);
        if (inputState.guessing==0) {
            addMinus_AST = (SandboxAST)astFactory.create(addMinus);
            astFactory.makeASTRoot(currentAST, addMinus_AST);
        }
        match(MINUS);
        break;
    }
    default:
    {
        throw new NoViableAltException(LT(1), getFilename());
    }
}
multiplicativeExpr();
if (inputState.guessing==0) {
    astFactory.addASTChild(currentAST, returnAST);
}
}
else {
    break _loop45;
}
} while (true);
}
if ( inputState.guessing==0 ) {
    if (!restriction.isEnabled(Restriction.ADDITION) && addPlus != null)
        reportError ("Addition statements (+)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
            addPlus.getLine(), addPlus.getColumn());

    if (!restriction.isEnabled(Restriction.SUBTRACTION) && addMinus != null)
        reportError ("Subtraction statements (-)", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
            addMinus.getLine(), addMinus.getColumn());

}
additiveExpr_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_23);
    } else {
        throw ex;
    }
}
returnAST = additiveExpr_AST;
}

/** (6.5.7) shift-expr: */
public final void shiftExpr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST shiftExpr_AST = null;
    Token lshift = null;
    SandboxAST lshift_AST = null;
    Token rshift = null;
    SandboxAST rshift_AST = null;

    try { // for error handling
        additiveExpr();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
    }
    _loop49:
    do {
        if ((LA(1)==LSHIFT||LA(1)==RSHIFT)) {
            {
                switch ( LA(1) ) {
                case LSHIFT:
                {

```

```

        lshift = LT(1);
        if (inputState.guessing==0) {
            lshift_AST = (SandboxAST)astFactory.create(lshift);
            astFactory.makeASTRoot(currentAST, lshift_AST);
        }
        match(LSHIFT);
        break;
    }
    case RSHIFT:
    {
        rshift = LT(1);
        if (inputState.guessing==0) {
            rshift_AST = (SandboxAST)astFactory.create(rshift);
            astFactory.makeASTRoot(currentAST, rshift_AST);
        }
        match(RSHIFT);
        break;
    }
    default:
    {
        throw new NoViableAltException(LT(1), getFilename());
    }
    }
    additiveExpr();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    }
    else {
        break _loop49;
    }
    }
    while (true);
    }
    if ( inputState.guessing==0 ) {

        if (!restriction.isEnabled(Restriction.LEFT) && lshift != null)
            reportError ("Left shift statements (<<)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
                lshift.getLine(), lshift.getColumn());

        if (!restriction.isEnabled(Restriction.RIGHT) && rshift != null)
            reportError ("Right shift statements (>>)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
                rshift.getLine(), rshift.getColumn());

    }
    shiftExpr_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_24);
    } else {
        throw ex;
    }
}
returnAST = shiftExpr_AST;
}
}

/** (6.5.8) relational-expr: */
public final void relationalExpr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST relationalExpr_AST = null;
    Token lt = null;
    SandboxAST lt_AST = null;
    Token gt = null;
    SandboxAST gt_AST = null;
    Token ltet = null;
    SandboxAST ltet_AST = null;
    Token gtet = null;
    SandboxAST gtet_AST = null;

```

```

try {      // for error handling
  shiftExpr();
  if (inputState.guessing==0) {
    astFactory.addASTChild(currentAST, returnAST);
  }
  {
  _loop53:
  do {
    if (((LA(1) >= LT && LA(1) <= GE))) {
      {
      switch ( LA(1) ) {
      case LT:
      {
        lt = LT(1);
        if (inputState.guessing==0) {
          lt_AST = (SandboxAST)astFactory.create(lt);
          astFactory.makeASTRoot(currentAST, lt_AST);
        }
        match(LT);
        break;
      }
      case GT:
      {
        gt = LT(1);
        if (inputState.guessing==0) {
          gt_AST = (SandboxAST)astFactory.create(gt);
          astFactory.makeASTRoot(currentAST, gt_AST);
        }
        match(GT);
        break;
      }
      case LE:
      {
        ltet = LT(1);
        if (inputState.guessing==0) {
          ltet_AST = (SandboxAST)astFactory.create(ltet);
          astFactory.makeASTRoot(currentAST, ltet_AST);
        }
        match(LE);
        break;
      }
      case GE:
      {
        gtet = LT(1);
        if (inputState.guessing==0) {
          gtet_AST = (SandboxAST)astFactory.create(gtet);
          astFactory.makeASTRoot(currentAST, gtet_AST);
        }
        match(GE);
        break;
      }
      default:
      {
        throw new NoViableAltException(LT(1), getFilename());
      }
      }
      shiftExpr();
      if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
      }
    }
    else {
      break _loop53;
    }
  } while (true);
}
if ( inputState.guessing==0 ) {

  if (!restriction.isEnabled(Restriction.LESS_THAN) && lt != null)
    reportError ("Less-than comparison statements (<)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
      lt.getLine(), lt.getColumn());
}

```

```

        if (!restriction.isEnabled(Restriction.LESS_THAN_EQTO) && ltet != null)
            reportError ("Less-than-or-equal-to comparison statements (<=)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
                ltet.getLine(), ltet.getColumn());

        if (!restriction.isEnabled(Restriction.GREATER_THAN) && gt != null)
            reportError ("Greater-than comparison statements (>)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
                gt.getLine(), gt.getColumn());

        if (!restriction.isEnabled(Restriction.GREATER_THAN_EQTO) && gtet != null)
            reportError ("Greater-than-or-equal-to comparison statements (>=)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
                gtet.getLine(), gtet.getColumn());
    }
    relationalExpr_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_25);
    } else {
        throw ex;
    }
}
returnAST = relationalExpr_AST;
}

/** (6.5.9) equality-expr: */
public final void equalityExpr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST equalityExpr_AST = null;
    Token eq = null;
    SandboxAST eq_AST = null;
    Token ne = null;
    SandboxAST ne_AST = null;

    try { // for error handling
        relationalExpr();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        {
        _loop57:
        do {
            if ((LA(1)==EQ||LA(1)==NE)) {
                {
                switch ( LA(1) ) {
                case EQ:
                {
                    eq = LT(1);
                    if (inputState.guessing==0) {
                        eq_AST = (SandboxAST)astFactory.create(eq);
                        astFactory.makeASTRoot(currentAST, eq_AST);
                    }
                    match(EQ);
                    break;
                }
                case NE:
                {
                    ne = LT(1);
                    if (inputState.guessing==0) {
                        ne_AST = (SandboxAST)astFactory.create(ne);
                        astFactory.makeASTRoot(currentAST, ne_AST);
                    }
                    match(NE);
                    break;
                }
                }
            }
            default:
            {
                throw new NoViableAltException(LT(1), getFilename());
            }
        }
        }
    }
}

```

```

    }
    relationalExpr();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
}
else {
    break _loop57;
}
} while (true);
}
if ( inputState.guessing==0 ) {

    if (!restriction.isEnabled(Restriction.EQUALITY) && eq != null)
        reportError ("Equality comparison statements (==)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
            eq.getLine(), eq.getColumn());

    if (!restriction.isEnabled(Restriction.INEQUALITY) && ne != null)
        reportError ("Inequality comparison statements (!=)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
            ne.getLine(), ne.getColumn());

}
equalityExpr_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_26);
    } else {
        throw ex;
    }
}
returnAST = equalityExpr_AST;
}
}

/** (6.5.13) logical-AND-expr: */
public final void logicalAndExpr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST logicalAndExpr_AST = null;
    Token land = null;
    SandboxAST land_AST = null;

    try { // for error handling
        equalityExpr();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
    }
    {
    _loop60:
    do {
        if ((LA(1)==LAND)) {
            land = LT(1);
            if (inputState.guessing==0) {
                land_AST = (SandboxAST)astFactory.create(land);
                astFactory.makeASTRoot(currentAST, land_AST);
            }
            match(LAND);
            equalityExpr();
            if (inputState.guessing==0) {
                astFactory.addASTChild(currentAST, returnAST);
            }
        }
        else {
            break _loop60;
        }
    } while (true);
}
if ( inputState.guessing==0 ) {

```

```

        if (!restriction.isEnabled(Restriction.AND) && land != null)
            reportError ("Logical AND statements (&&)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
                land.getLine(), land.getColumn());
    }
    logicalAndExpr_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_27);
    } else {
        throw ex;
    }
}
}
returnAST = logicalAndExpr_AST;
}

/** (6.5.14) logical-OR-expr: */
public final void logicalOrExpr() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST logicalOrExpr_AST = null;
    Token lor = null;
    SandboxAST lor_AST = null;

    try { // for error handling
        logicalAndExpr();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        {
        _loop63:
        do {
            if ((LA(1)==LOR)) {
                lor = LT(1);
                if (inputState.guessing==0) {
                    lor_AST = (SandboxAST)astFactory.create(lor);
                    astFactory.makeASTRoot(currentAST, lor_AST);
                }
                match(LOR);
                logicalAndExpr();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
            }
            else {
                break _loop63;
            }
        } while (true);
        }
        if ( inputState.guessing==0 ) {

            if (!restriction.isEnabled(Restriction.OR) && lor != null)
                reportError ("Logical OR statements (||)", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
                    lor.getLine(), lor.getColumn());

        }
        logicalOrExpr_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_20);
        } else {
            throw ex;
        }
    }
}
returnAST = logicalOrExpr_AST;
}

```



```

        astFactory.addASTChild(currentAST, returnAST);
    }
}
else {
    if ( _cnt75>=1 ) { break _loop75; } else {throw new NoViableAltException(LT(1), getFilename());}
}

_cnt75++;
} while (true);
}
initDeclaratorList();
if (inputState.guessing==0) {
    astFactory.addASTChild(currentAST, returnAST);
}
SandboxAST tmp55_AST = null;
tmp55_AST = (SandboxAST)astFactory.create(LT(1));
match(SEMI);
if ( inputState.guessing==0 ) {
    declaration_AST = (SandboxAST)currentAST.root;
    declaration_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
        SandboxAST)astFactory.create(DECLARATION,"DECLARATION")).add(declaration_AST));
    currentAST.root = declaration_AST;
    currentAST.child = declaration_AST!=null &&declaration_AST.getFirstChild()!=null ?
        declaration_AST.getFirstChild() : declaration_AST;
    currentAST.advanceChildToEnd();
}
declaration_AST = (SandboxAST)currentAST.root;
break;
}
case TYPEDEF:
{
    typedef = LT(1);
    if (inputState.guessing==0) {
        typedef_AST = (SandboxAST)astFactory.create(typedef);
        astFactory.addASTChild(currentAST, typedef_AST);
    }
    match(TYPEDEF);
    if ( inputState.guessing==0 ) {
        inTypedef = true;
    }
    {
    }
    int _cnt77=0;
_loop77:
do {
    if ((_tokenSet_29.member(LA(1)))) {
        declarationSpecifier();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
    }
    else {
        if ( _cnt77>=1 ) { break _loop77; } else {throw new NoViableAltException(LT(1), getFilename());}
    }
}
_cnt77++;
} while (true);
}
initDeclaratorList();
if (inputState.guessing==0) {
    astFactory.addASTChild(currentAST, returnAST);
}
SandboxAST tmp56_AST = null;
tmp56_AST = (SandboxAST)astFactory.create(LT(1));
match(SEMI);
if ( inputState.guessing==0 ) {
    declaration_AST = (SandboxAST)currentAST.root;

    if (!restriction.isEnabled(Restriction.TYPEDEF) && typedef != null)
        reportError ("Typedef statements", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
            typedef.getLine(), typedef.getColumn());
    declaration_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
        SandboxAST)astFactory.create(DECLARATION,"DECLARATION")).add(declaration_AST));
    inTypedef = false;
}

```



```

    }
    break;
}
case ENUM:
case STRUCT:
case TYPENAME:
{
    break;
}
default:
{
    throw new NoViableAltException(LT(1), getFilename());
}
}
}
typeSpecifier();
if (inputState.guessing==0) {
    astFactory.addASTChild(currentAST, returnAST);
}
{
switch ( LA(1)) {
case AMPERSAND:
{
    ptrOperator();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    break;
}
case CONST:
case ENUM:
case STRUCT:
case TYPENAME:
case RPAREN:
case SEMI:
case COMMA:
case IDENTIFIER:
{
    break;
}
default:
{
    throw new NoViableAltException(LT(1), getFilename());
}
}
}
declarationSpecifier_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_32);
    } else {
        throw ex;
    }
}
returnAST = declarationSpecifier_AST;
}
}

/**
 * (6.7) init-declarator-list: comma-separated list of the names of
 * variables to declare.
 */
public final void initDeclaratorList() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST initDeclaratorList_AST = null;

    try { // for error handling
        {
            switch ( LA(1)) {
                case IDENTIFIER:

```

```

    {
        initDeclarator();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        {
        _loop94:
        do {
            if ((LA(1)==COMMA)) {
                SandboxAST tmp57_AST = null;
                tmp57_AST = (SandboxAST)astFactory.create(LT(1));
                match(COMMA);
                initDeclarator();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
            }
            else {
                break _loop94;
            }
        } while (true);
        }
        break;
    }
    case SEMI:
    {
        break;
    }
    default:
    {
        throw new NoViableAltException(LT(1), getFilename());
    }
    }
    }
    initDeclaratorList_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_31);
    }
    else {
        throw ex;
    }
}
returnAST = initDeclaratorList_AST;
}

public final void usingDirective() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST usingDirective_AST = null;

    try { // for error handling
        SandboxAST tmp58_AST = null;
        tmp58_AST = (SandboxAST)astFactory.create(LT(1));
        match(USING);
        SandboxAST tmp59_AST = null;
        tmp59_AST = (SandboxAST)astFactory.create(LT(1));
        match(NAMESPACE);
        nestedNameSpecifier();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        if ( inputState.guessing==0 ) {
            usingDirective_AST = (SandboxAST)currentAST.root;
            usingDirective_AST = (SandboxAST)astFactory.make( ( new ASTArray(2) ).add((
                SandboxAST)astFactory.create(USING_DIRECTIVE, "USING_DIRECTIVE") ).add(usingDirective_AST));
            currentAST.root = usingDirective_AST;
            currentAST.child = usingDirective_AST!=null &&usingDirective_AST.getFirstChild()!=null ?
                usingDirective_AST.getFirstChild() : usingDirective_AST;
            currentAST.advanceChildToEnd();
        }
    }
}

```

```

    }
    usingDirective_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_31);
    } else {
        throw ex;
    }
}
returnAST = usingDirective_AST;
}

public final void nestedNameSpecifier() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST nestedNameSpecifier_AST = null;

    try {        // for error handling
        {
            _loop82:
            do {
                if ((LA(1)==IDENTIFIER)) {
                    SandboxAST tmp60_AST = null;
                    if (inputState.guessing==0) {
                        tmp60_AST = (SandboxAST)astFactory.create(LT(1));
                        astFactory.addASTChild(currentAST, tmp60_AST);
                    }
                    match(IDENTIFIER);
                }
                else {
                    break _loop82;
                }
            } while (true);
        }
        nestedNameSpecifier_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_31);
        } else {
            throw ex;
        }
    }
    returnAST = nestedNameSpecifier_AST;
}

public final void typeModifiers() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST typeModifiers_AST = null;

    try {        // for error handling
        {
            int _cnt89=0;
            _loop89:
            do {
                if ((LA(1)==CONST)) {
                    typeModifier();
                    if (inputState.guessing==0) {
                        astFactory.addASTChild(currentAST, returnAST);
                    }
                }
                else {
                    if ( _cnt89>=1 ) { break _loop89; } else {throw new NoViableAltException(LT(1), getFilename());}
                }
            }
        }
    }
}

```

```

        _cnt89++;
    } while (true);
    }
    if ( inputState.guessing==0 ) {
        typeModifiers_AST = (SandboxAST)currentAST.root;
        typeModifiers_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
            SandboxAST)astFactory.create(TYPE_MODIFIER,"TYPE_MODIFIER")).add(typeModifiers_AST));
        currentAST.root = typeModifiers_AST;
        currentAST.child = typeModifiers_AST!=null &&typeModifiers_AST.getFirstChild()!=null ?
            typeModifiers_AST.getFirstChild() : typeModifiers_AST;
        currentAST.advanceChildToEnd();
    }
    typeModifiers_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_33);
    } else {
        throw ex;
    }
}
}
returnAST = typeModifiers_AST;
}

public final void ptrOperator() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST ptrOperator_AST = null;

    try { // for error handling
        SandboxAST tmp61_AST = null;
        if (inputState.guessing==0) {
            tmp61_AST = (SandboxAST)astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp61_AST);
        }
        match(AMPERSAND);
        ptrOperator_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_32);
        } else {
            throw ex;
        }
    }
}
returnAST = ptrOperator_AST;
}

public final void typeModifier() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST typeModifier_AST = null;
    Token c = null;
    SandboxAST c_AST = null;

    try { // for error handling
        c = LT(1);
        if (inputState.guessing==0) {
            c_AST = (SandboxAST)astFactory.create(c);
            astFactory.addASTChild(currentAST, c_AST);
        }
        match(CONST);
        if ( inputState.guessing==0 ) {

            if (!restriction.isEnabled(Restriction.CONST) && c != null)
                reportError ("Constant identifiers", ErrorCodeTable.ARE_DISABLED_2_PARAMS, c.getLine(), c.getColumn());

        }
    }
}

```

```

        typeModifier_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_29);
        } else {
            throw ex;
        }
    }
}
returnAST = typeModifier_AST;
}

/** (6.7) init-declarator: allows '=' to be used in variable declarations. */
public final void initDeclarator() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST initDeclarator_AST = null;
    SandboxAST d_AST = null;

    try { // for error handling
        declarator();
        if (inputState.guessing==0) {
            d_AST = (SandboxAST)returnAST;
            astFactory.addASTChild(currentAST, returnAST);
        }
        {
            switch ( LA(1)) {
            case ASSIGN:
            {
                SandboxAST tmp62_AST = null;
                tmp62_AST = (SandboxAST)astFactory.create(LT(1));
                match(ASSIGN);
                initializer();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
                break;
            }
            case SEMI:
            case COMMA:
            {
                break;
            }
            default:
            {
                throw new NoViableAltException(LT(1), getFilename());
            }
        }
    }
    if ( inputState.guessing==0 ) {
        initDeclarator_AST = (SandboxAST)currentAST.root;
        if (inTypedef)
            addType (d_AST.getText());
            initDeclarator_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
                SandboxAST)astFactory.create(DECLARATOR,"DECLARATOR").add(initDeclarator_AST));
        currentAST.root = initDeclarator_AST;
        currentAST.child = initDeclarator_AST!=null &&initDeclarator_AST.getFirstChild()!=null ?
            initDeclarator_AST.getFirstChild() : initDeclarator_AST;
        currentAST.advanceChildToEnd();
    }
    initDeclarator_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_34);
    } else {
        throw ex;
    }
}
}

```

```

    returnAST = initDeclarator_AST;
}

/** (6.7.5) declarator: is direct-declarator; pointers aren't implemented */
public final void declarator() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST declarator_AST = null;

    try {        // for error handling
        boolean synPredMatched122 = false;
        if (((LA(1)==IDENTIFIER) && (LA(2)==LPAREN))) {
            int _m122 = mark();
            synPredMatched122 = true;
            inputState.guessing++;
            try {
                {
                    match(IDENTIFIER);
                    match(LPAREN);
                }
            }
            catch (RecognitionException pe) {
                synPredMatched122 = false;
            }
            rewind(_m122);
            inputState.guessing--;
        }
        if ( synPredMatched122 ) {
            SandboxAST tmp63_AST = null;
            if (inputState.guessing==0) {
                tmp63_AST = (SandboxAST)astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp63_AST);
            }
            match(IDENTIFIER);
            SandboxAST tmp64_AST = null;
            tmp64_AST = (SandboxAST)astFactory.create(LT(1));
            match(LPAREN);
            parameterList();
            if (inputState.guessing==0) {
                astFactory.addASTChild(currentAST, returnAST);
            }
            SandboxAST tmp65_AST = null;
            tmp65_AST = (SandboxAST)astFactory.create(LT(1));
            match(RPAREN);
            declarator_AST = (SandboxAST)currentAST.root;
        }
        else if ((LA(1)==IDENTIFIER) && (_tokenSet_35.member(LA(2)))) {
            SandboxAST tmp66_AST = null;
            if (inputState.guessing==0) {
                tmp66_AST = (SandboxAST)astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp66_AST);
            }
            match(IDENTIFIER);
            declarator_AST = (SandboxAST)currentAST.root;
        }
        else if ((LA(1)==IDENTIFIER) && (LA(2)==LBRACKET)) {
            SandboxAST tmp67_AST = null;
            if (inputState.guessing==0) {
                tmp67_AST = (SandboxAST)astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp67_AST);
            }
            match(IDENTIFIER);
            {
                int _cnt124=0;
                _loop124:
                do {
                    if ((LA(1)==LBRACKET)) {
                        arrayDesignator();
                        if (inputState.guessing==0) {
                            astFactory.addASTChild(currentAST, returnAST);
                        }
                    }
                }
                else {

```

```

        if ( _cnt124>=1 ) { break _loop124; } else {throw new NoViableAltException(LT(1), getFilename());}
    }

    _cnt124++;
    } while (true);
    }
    if ( inputState.guessing==0 ) {
        declarator_AST = (SandboxAST)currentAST.root;
        declarator_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
            SandboxAST)astFactory.create(ARRAY_DECLARATOR,"ARRAY_DECLARATOR")).add(declarator_AST));
        currentAST.root = declarator_AST;
        currentAST.child = declarator_AST!=null &&declarator_AST.getFirstChild()!=null ?
            declarator_AST.getFirstChild() : declarator_AST;
        currentAST.advanceChildToEnd();
    }
    declarator_AST = (SandboxAST)currentAST.root;
    }
    else {
        throw new NoViableAltException(LT(1), getFilename());
    }
}
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_35);
    } else {
        throw ex;
    }
}
}
returnAST = declarator_AST;
}

/** (6.7.8) initializer: used to initialize variables. */
public final void initializer() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST initializer_AST = null;

    try { // for error handling
        switch ( LA(1) ) {
        case FLOATING_CONSTANT:
        case CONST_CAST:
        case ENUM:
        case FALSE:
        case REINTERPRET_CAST:
        case SIZEOF:
        case STATIC_CAST:
        case STRUCT:
        case TYPENAME:
        case TRUE:
        case LPAREN:
        case INCR:
        case DECR:
        case PLUS:
        case MINUS:
        case TILDE:
        case BANG:
        case IDENTIFIER:
        case INTEGER_CONSTANT:
        case CHAR_CONSTANT:
        case STRING_LITERAL:
        {
            assignmentExpr();
            if (inputState.guessing==0) {
                astFactory.addASTChild(currentAST, returnAST);
            }
            initializer_AST = (SandboxAST)currentAST.root;
            break;
        }
        case LBRACE:
        {

```

```

SandboxAST tmp68_AST = null;
tmp68_AST = (SandboxAST)astFactory.create(LT(1));
match(LBRACE);
initializerList();
if (inputState.guessing==0) {
    astFactory.addASTChild(currentAST, returnAST);
}
}
switch ( LA(1)) {
case COMMA:
{
    SandboxAST tmp69_AST = null;
    tmp69_AST = (SandboxAST)astFactory.create(LT(1));
    match(COMMA);
    break;
}
case RBRACE:
{
    break;
}
default:
{
    throw new NoViableAltException(LT(1), getFilename());
}
}
SandboxAST tmp70_AST = null;
tmp70_AST = (SandboxAST)astFactory.create(LT(1));
match(RBRACE);
if ( inputState.guessing==0 ) {
    initializer_AST = (SandboxAST)currentAST.root;
    initializer_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
        SandboxAST)astFactory.create(INITIALIZER,"INITIALIZER")).add(initializer_AST));
    currentAST.root = initializer_AST;
    currentAST.child = initializer_AST!=null &&initializer_AST.getFirstChild()!=null ?
        initializer_AST.getFirstChild() : initializer_AST;
    currentAST.advanceChildToEnd();
}
initializer_AST = (SandboxAST)currentAST.root;
break;
}
default:
{
    throw new NoViableAltException(LT(1), getFilename());
}
}
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_36);
    } else {
        throw ex;
    }
}
returnAST = initializer_AST;
}
}

/**
 * (6.7.2.1) struct-or-union-specifier: structure declarations.
 *
 * We're not implementing unions (yet).
 *
 * There's too much duplicated code here. However, it's needed in order to
 * allow two-token lookahead, which is needed to prevent the decl-struct
 * issue.
 */
public final void structSpecifier() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST structSpecifier_AST = null;
    Token i = null;

```

```

SandboxAST i_AST = null;

try {      // for error handling
  if ((LA(1)==STRUCT) && (LA(2)==LBRACE)) {
    SandboxAST tmp71_AST = null;
    if (inputState.guessing==0) {
      tmp71_AST = (SandboxAST)astFactory.create(LT(1));
      astFactory.addASTChild(currentAST, tmp71_AST);
    }
    match(STRUCT);
    SandboxAST tmp72_AST = null;
    tmp72_AST = (SandboxAST)astFactory.create(LT(1));
    match(LBRACE);
    structDeclarationList();
    if (inputState.guessing==0) {
      astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp73_AST = null;
    tmp73_AST = (SandboxAST)astFactory.create(LT(1));
    match(RBRACE);
    if ( inputState.guessing==0 ) {
      structSpecifier_AST = (SandboxAST)currentAST.root;
      structSpecifier_AST = (SandboxAST)astFactory.make( new ASTArray(2)).add((
        SandboxAST)astFactory.create(STRUCT_DECLARATION,"STRUCT_DECLARATION")).add(structSpecifier_AST);
      currentAST.root = structSpecifier_AST;
      currentAST.child = structSpecifier_AST!=null &&structSpecifier_AST.getFirstChild()!=null ?
        structSpecifier_AST.getFirstChild() : structSpecifier_AST;
      currentAST.advanceChildToEnd();
    }
    structSpecifier_AST = (SandboxAST)currentAST.root;
  }
} else if ((LA(1)==STRUCT) && (LA(2)==IDENTIFIER)) {
  SandboxAST tmp74_AST = null;
  if (inputState.guessing==0) {
    tmp74_AST = (SandboxAST)astFactory.create(LT(1));
    astFactory.addASTChild(currentAST, tmp74_AST);
  }
  match(STRUCT);
  i = LT(1);
  if (inputState.guessing==0) {
    i_AST = (SandboxAST)astFactory.create(i);
    astFactory.addASTChild(currentAST, i_AST);
  }
  match(IDENTIFIER);
  if ( inputState.guessing==0 ) {
    addType (i_AST.getText());
    if (!restriction.isEnabled(Restriction.STRUCT) && i != null)
      reportError ("Creating user-defined structs", ErrorCodeTable.ARE_DISABLED_3_PARAMS,
        i.getLine(), i.getColumn());
  }
}
{
switch ( LA(1) ) {
case LBRACE:
{
  SandboxAST tmp75_AST = null;
  tmp75_AST = (SandboxAST)astFactory.create(LT(1));
  match(LBRACE);
  structDeclarationList();
  if (inputState.guessing==0) {
    astFactory.addASTChild(currentAST, returnAST);
  }
  SandboxAST tmp76_AST = null;
  tmp76_AST = (SandboxAST)astFactory.create(LT(1));
  match(RBRACE);
  break;
}
case EOF:
case CONST:
case ENUM:
case STRUCT:
case TYPENAME:
case LPAREN:
case RPAREN:

```

```

case AMPERSAND:
case GT:
case SEMI:
case COMMA:
case IDENTIFIER:
{
    break;
}
default:
{
    throw new NoViableAltException(LT(1), getFilename());
}
}
}
if ( inputState.guessing==0 ) {
    structSpecifier_AST = (SandboxAST)currentAST.root;
    structSpecifier_AST = (SandboxAST)astFactory.make( new ASTArray(2)).add((
        SandboxAST)astFactory.create(STRUCT_DECLARATION,"STRUCT_DECLARATION")).add(structSpecifier_AST);
    currentAST.root = structSpecifier_AST;
    currentAST.child = structSpecifier_AST!=null &&structSpecifier_AST.getFirstChild()!=null ?
        structSpecifier_AST.getFirstChild() : structSpecifier_AST;
    currentAST.advanceChildToEnd();
}
structSpecifier_AST = (SandboxAST)currentAST.root;
}
else if ((LA(1)==STRUCT) && (LA(2)==TYPENAME)) {
    SandboxAST tmp77_AST = null;
    if (inputState.guessing==0) {
        tmp77_AST = (SandboxAST)astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp77_AST);
    }
    match(STRUCT);
    SandboxAST tmp78_AST = null;
    if (inputState.guessing==0) {
        tmp78_AST = (SandboxAST)astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp78_AST);
    }
    match(TYPENAME);
    {
    switch ( LA(1) ) {
    case LBRACE:
    {
        SandboxAST tmp79_AST = null;
        tmp79_AST = (SandboxAST)astFactory.create(LT(1));
        match(LBRACE);
        structDeclarationList();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        SandboxAST tmp80_AST = null;
        tmp80_AST = (SandboxAST)astFactory.create(LT(1));
        match(RBRACE);
        break;
    }
    case EOF:
    case CONST:
    case ENUM:
    case STRUCT:
    case TYPENAME:
    case LPAREN:
    case RPAREN:
    case AMPERSAND:
    case GT:
    case SEMI:
    case COMMA:
    case IDENTIFIER:
    {
        break;
    }
    default:
    {
        throw new NoViableAltException(LT(1), getFilename());
    }
    }
}
}
}

```

```

    }
    if ( inputState.guessing==0 ) {
        structSpecifier_AST = (SandboxAST)currentAST.root;
        structSpecifier_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
            SandboxAST)astFactory.create(STRUCT_DECLARATION,"STRUCT_DECLARATION")).add(structSpecifier_AST));
        currentAST.root = structSpecifier_AST;
        currentAST.child = structSpecifier_AST!=null &&structSpecifier_AST.getFirstChild()!=null ?
            structSpecifier_AST.getFirstChild() : structSpecifier_AST;
        currentAST.advanceChildToEnd();
    }
    structSpecifier_AST = (SandboxAST)currentAST.root;
}
else {
    throw new NoViableAltException(LT(1), getFilename());
}
}
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_5);
    } else {
        throw ex;
    }
}
}
returnAST = structSpecifier_AST;
}
}

/** (6.7.2.2) enum-specifier: the declaration format for enumerations. */
public final void enumSpecifier() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST enumSpecifier_AST = null;
    Token e = null;
    SandboxAST e_AST = null;

    try { // for error handling
        e = LT(1);
        if (inputState.guessing==0) {
            e_AST = (SandboxAST)astFactory.create(e);
            astFactory.addASTChild(currentAST, e_AST);
        }
        match(ENUM);
        {
        switch ( LA(1) ) {
        case IDENTIFIER:
            {
                SandboxAST tmp81_AST = null;
                if (inputState.guessing==0) {
                    tmp81_AST = (SandboxAST)astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp81_AST);
                }
                match(IDENTIFIER);
                break;
            }
        case LBRACE:
            {
                break;
            }
        default:
            {
                throw new NoViableAltException(LT(1), getFilename());
            }
        }
        }
        SandboxAST tmp82_AST = null;
        tmp82_AST = (SandboxAST)astFactory.create(LT(1));
        match(LBRACE);
        enumeratorList();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
    }
}

```

```

SandboxAST tmp83_AST = null;
tmp83_AST = (SandboxAST)astFactory.create(LT(1));
match(RBRACE);
if ( inputState.guessing==0 ) {

    if (!restriction.isEnabled(Restriction.ENUM) && e != null)
        reportError ("Using enumerated types", ErrorCodeTable.ARE_DISABLED_3_PARAMS, e.getLine(), e.getColumn());

}
enumSpecifier_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_5);
    } else {
        throw ex;
    }
}
returnAST = enumSpecifier_AST;
}

/** (6.7.2.1) struct-declaration-list: declare the members of a struct. */
public final void structDeclarationList() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST structDeclarationList_AST = null;

    try {        // for error handling
    {
        _loop103:
        do {
            if ((_tokenSet_29.member(LA(1)))) {
                structDeclaration();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
            }
            else {
                break _loop103;
            }
        } while (true);
    }
    if ( inputState.guessing==0 ) {
        structDeclarationList_AST = (SandboxAST)currentAST.root;
        structDeclarationList_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
            SandboxAST)astFactory.create(STRUCT_MEMBERS,"STRUCT_MEMBERS")).add(structDeclarationList_AST));
        currentAST.root = structDeclarationList_AST;
        currentAST.child = structDeclarationList_AST!=null &&structDeclarationList_AST.getFirstChild()!=null ?
            structDeclarationList_AST.getFirstChild() : structDeclarationList_AST;
        currentAST.advanceChildToEnd();
    }
    structDeclarationList_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_37);
    } else {
        throw ex;
    }
}
returnAST = structDeclarationList_AST;
}

/**
 * (6.7.2.1) struct-declaration: declare an individual member of a structure
 */
public final void structDeclaration() throws RecognitionException, TokenStreamException {

```

```

returnAST = null;
ASTPair currentAST = new ASTPair();
SandboxAST structDeclaration_AST = null;

try {      // for error handling
    specifierQualifierList();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    structDeclaratorList();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp84_AST = null;
    tmp84_AST = (SandboxAST)astFactory.create(LT(1));
    match(SEMI);
    structDeclaration_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_38);
    } else {
        throw ex;
    }
}
returnAST = structDeclaration_AST;
}

/** (6.7.2.1) specifier-qualifier-list: */
public final void specifierQualifierList() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
SandboxAST specifierQualifierList_AST = null;

try {      // for error handling
    {
        int _cnt107=0;
        _loop107:
        do {
            switch ( LA(1)) {
            case ENUM:
            case STRUCT:
            case TYPENAME:
            {
                typeSpecifier();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
                break;
            }
            case CONST:
            {
                SandboxAST tmp85_AST = null;
                if (inputState.guessing==0) {
                    tmp85_AST = (SandboxAST)astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp85_AST);
                }
                match(CONST);
                break;
            }
            default:
            {
                if ( _cnt107>=1 ) { break _loop107; } else {throw new NoViableAltException(LT(1), getFilename());}
            }
            }
            _cnt107++;
        } while (true);
    }
    specifierQualifierList_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {

```

```

        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_39);
        } else {
            throw ex;
        }
    }
}
returnAST = specifierQualifierList_AST;
}

/** (6.7.2.1) struct-declaration-list: declares names of members of a struct. */
public final void structDeclaratorList() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST structDeclaratorList_AST = null;

    try {        // for error handling
    {
        switch ( LA(1)) {
        case IDENTIFIER:
        {
            structDeclarator();
            if (inputState.guessing==0) {
                astFactory.addASTChild(currentAST, returnAST);
            }
        }
        _loop111:
        do {
            if ((LA(1)==COMMA)) {
                SandboxAST tmp86_AST = null;
                tmp86_AST = (SandboxAST)astFactory.create(LT(1));
                match(COMMA);
                structDeclarator();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
            }
            else {
                break _loop111;
            }
        } while (true);
        }
        break;
    }
    case SEMI:
    {
        break;
    }
    default:
    {
        throw new NoViableAltException(LT(1), getFilename());
    }
    }
    structDeclaratorList_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_31);
    } else {
        throw ex;
    }
}
returnAST = structDeclaratorList_AST;
}

/**
 * (6.7.2.1) struct-declarator: declares a single member.
 *

```

```

* Also used to declare bitfields (unimplemented).
*/
public final void structDeclarator() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST structDeclarator_AST = null;

    try {        // for error handling
        declarator();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        structDeclarator_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_34);
        } else {
            throw ex;
        }
    }
    returnAST = structDeclarator_AST;
}

/** (6.7.2.2) enumerator-list: list of the "members" of an enumeration. */
public final void enumeratorList() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST enumeratorList_AST = null;

    try {        // for error handling
        enumerator();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        {
        _loop117:
        do {
            if ((LA(1)==COMMA)) {
                SandboxAST tmp87_AST = null;
                tmp87_AST = (SandboxAST)astFactory.create(LT(1));
                match(COMMA);
                enumerator();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
            }
            else {
                break _loop117;
            }
        } while (true);
        if ( inputState.guessing==0 ) {
            enumeratorList_AST = (SandboxAST)currentAST.root;
            enumeratorList_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
                SandboxAST)astFactory.create(ENUMERATOR_LIST,"ENUMERATOR_LIST")).add(enumeratorList_AST));
            currentAST.root = enumeratorList_AST;
            currentAST.child = enumeratorList_AST!=null &&enumeratorList_AST.getFirstChild()!=null ?
                enumeratorList_AST.getFirstChild() : enumeratorList_AST;
            currentAST.advanceChildToEnd();
        }
        enumeratorList_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_37);
        } else {

```

```

        throw ex;
    }
}
returnAST = enumeratorList_AST;
}

/**
 * (6.7.2.2) enumerator: actual "member(s)" of an enumeration.
 *
 * Consists of either a name, or a name and a value:
 *
 * enum foo
 * {
 *     bar,      // name
 *     baz = 1   // name & value
 * }
 */
public final void enumerator() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST enumerator_AST = null;

    try {        // for error handling
        SandboxAST tmp88_AST = null;
        if (inputState.guessing==0) {
            tmp88_AST = (SandboxAST)astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp88_AST);
        }
        match(IDENTIFIER);
        {
        switch ( LA(1) ) {
        case ASSIGN:
            {
                SandboxAST tmp89_AST = null;
                tmp89_AST = (SandboxAST)astFactory.create(LT(1));
                match(ASSIGN);
                SandboxAST tmp90_AST = null;
                if (inputState.guessing==0) {
                    tmp90_AST = (SandboxAST)astFactory.create(LT(1));
                    astFactory.addASTChild(currentAST, tmp90_AST);
                }
                match(INTEGER_CONSTANT);
                break;
            }
        case RBRACE:
        case COMMA:
            {
                break;
            }
        default:
            {
                throw new NoViableAltException(LT(1), getFilename());
            }
        }
        }
        if ( inputState.guessing==0 ) {
            enumerator_AST = (SandboxAST)currentAST.root;
            enumerator_AST = (SandboxAST)astFactory.make( ( new ASTArray(2) ).add((
                SandboxAST)astFactory.create(DECLARATOR,"DECLARATOR")).add(enumerator_AST));
            currentAST.root = enumerator_AST;
            currentAST.child = enumerator_AST!=null &&enumerator_AST.getFirstChild()!=null ?
                enumerator_AST.getFirstChild() : enumerator_AST;
            currentAST.advanceChildToEnd();
        }
        enumerator_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_40);
        }
        else {
            throw ex;
        }
    }
}

```

```

    }
  }
  returnAST = enumerator_AST;
}

/** (6.7.5) parameter-list: list of parameters to a function. */
public final void parameterList() throws RecognitionException, TokenStreamException {

  returnAST = null;
  ASTPair currentAST = new ASTPair();
  SandboxAST parameterList_AST = null;

  try {      // for error handling
    parameterDeclarations();
    if (inputState.guessing==0) {
      astFactory.addASTChild(currentAST, returnAST);
    }
    if ( inputState.guessing==0 ) {
      parameterList_AST = (SandboxAST)currentAST.root;
      parameterList_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
        SandboxAST)astFactory.create(PARAMETER_LIST,"PARAMETER_LIST")).add(parameterList_AST));
      currentAST.root = parameterList_AST;
      currentAST.child = parameterList_AST!=null &&parameterList_AST.getFirstChild()!=null ?
        parameterList_AST.getFirstChild() : parameterList_AST;
      currentAST.advanceChildToEnd();
    }
    parameterList_AST = (SandboxAST)currentAST.root;
  }
  catch (RecognitionException ex) {
    if (inputState.guessing==0) {
      reportError(ex);
      consume();
      consumeUntil(_tokenSet_6);
    } else {
      throw ex;
    }
  }
  returnAST = parameterList_AST;
}

public final void arrayDesignator() throws RecognitionException, TokenStreamException {

  returnAST = null;
  ASTPair currentAST = new ASTPair();
  SandboxAST arrayDesignator_AST = null;

  try {      // for error handling
    SandboxAST tmp91_AST = null;
    tmp91_AST = (SandboxAST)astFactory.create(LT(1));
    match(LBRACKET);
    {
      switch ( LA(1) ) {
      case FLOATING_CONSTANT:
      case CONST_CAST:
      case ENUM:
      case FALSE:
      case REINTERPRET_CAST:
      case SIZEOF:
      case STATIC_CAST:
      case STRUCT:
      case TYPENAME:
      case TRUE:
      case LPAREN:
      case INCR:
      case DECR:
      case PLUS:
      case MINUS:
      case TILDE:
      case BANG:
      case IDENTIFIER:
      case INTEGER_CONSTANT:
      case CHAR_CONSTANT:
      case STRING_LITERAL:
      {

```

```

        constantExpr();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        break;
    }
    case RBRACKET:
    {
        break;
    }
    default:
    {
        throw new NoViableAltException(LT(1), getFilename());
    }
}
SandboxAST tmp92_AST = null;
tmp92_AST = (SandboxAST)astFactory.create(LT(1));
match(RBRACKET);
arrayDesignator_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_41);
    } else {
        throw ex;
    }
}
returnAST = arrayDesignator_AST;
}

/** list of parameters to a function. */
public final void parameterDeclarations() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST parameterDeclarations_AST = null;

    try {        // for error handling
        parameterDeclaration();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
    }
    {
        _loop128:
        do {
            if ((LA(1)==COMMA)) {
                SandboxAST tmp93_AST = null;
                tmp93_AST = (SandboxAST)astFactory.create(LT(1));
                match(COMMA);
                parameterDeclaration();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
            }
            else {
                break _loop128;
            }
        } while (true);
    }
    parameterDeclarations_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_6);
    } else {
        throw ex;
    }
}
}

```

```

    returnAST = parameterDeclarations_AST;
}

/** (6.7.5) parameter-declaration: parameters to a function. */
public final void parameterDeclaration() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST parameterDeclaration_AST = null;

    try { // for error handling
        {
            int _cnt131=0;
            _loop131:
            do {
                if ((_tokenSet_29.member(LA(1)))) {
                    declarationSpecifier();
                    if (inputState.guessing==0) {
                        astFactory.addASTChild(currentAST, returnAST);
                    }
                }
                else {
                    if ( _cnt131>=1 ) { break _loop131; } else {throw new NoViableAltException(LT(1), getFilename());}
                }
            } while (true);
            {
                switch ( LA(1) ) {
                case IDENTIFIER:
                    {
                        declarator();
                        if (inputState.guessing==0) {
                            astFactory.addASTChild(currentAST, returnAST);
                        }
                    }
                    break;
                case RPAREN:
                case COMMA:
                    {
                        break;
                    }
                default:
                    {
                        throw new NoViableAltException(LT(1), getFilename());
                    }
                }
            }
            if ( inputState.guessing==0 ) {
                parameterDeclaration_AST = (SandboxAST)currentAST.root;
                parameterDeclaration_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
                    SandboxAST)astFactory.create(PARAMETER,"PARAMETER")).add(parameterDeclaration_AST));
                currentAST.root = parameterDeclaration_AST;
                currentAST.child = parameterDeclaration_AST!=null &&parameterDeclaration_AST.getFirstChild()!=null ?
                    parameterDeclaration_AST.getFirstChild() : parameterDeclaration_AST;
                currentAST.advanceChildToEnd();
            }
            parameterDeclaration_AST = (SandboxAST)currentAST.root;
        }
        catch (RecognitionException ex) {
            if (inputState.guessing==0) {
                reportError(ex);
                consume();
                consumeUntil(_tokenSet_42);
            }
            else {
                throw ex;
            }
        }
    }
    returnAST = parameterDeclaration_AST;
}

/** (6.7.5) identifier-list: list of identifiers (as used in function calls) */
public final void identifierList() throws RecognitionException, TokenStreamException {

```

```

returnAST = null;
ASTPair currentAST = new ASTPair();
SandboxAST identifierList_AST = null;

try { // for error handling
    SandboxAST tmp94_AST = null;
    if (inputState.guessing==0) {
        tmp94_AST = (SandboxAST)astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp94_AST);
    }
    match(IDENTIFIER);
    {
    _loop135:
    do {
        if ((LA(1)==COMMA)) {
            SandboxAST tmp95_AST = null;
            tmp95_AST = (SandboxAST)astFactory.create(LT(1));
            match(COMMA);
            SandboxAST tmp96_AST = null;
            if (inputState.guessing==0) {
                tmp96_AST = (SandboxAST)astFactory.create(LT(1));
                astFactory.addASTChild(currentAST, tmp96_AST);
            }
            match(IDENTIFIER);
        }
        else {
            break _loop135;
        }
    } while (true);
    }
    identifierList_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_0);
    } else {
        throw ex;
    }
}
returnAST = identifierList_AST;
}

/** (6.7.6) type-name: allows declaring the name of a type. */
public final void typeName() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST typeName_AST = null;

    try { // for error handling
        specifierQualifierList();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        typeName_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_0);
        } else {
            throw ex;
        }
    }
}
returnAST = typeName_AST;
}

/** (6.7.7) typedef-name: introduces a new type. */
public final void typedefName() throws RecognitionException, TokenStreamException {

```

```

returnAST = null;
ASTPair currentAST = new ASTPair();
SandboxAST typedefName_AST = null;

try {      // for error handling
    SandboxAST tmp97_AST = null;
    if (inputState.guessing==0) {
        tmp97_AST = (SandboxAST)astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp97_AST);
    }
    match(IDENTIFIER);
    typedefName_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_0);
    } else {
        throw ex;
    }
}
returnAST = typedefName_AST;
}

/**
 * (6.7.8) initializer-list: used when initializing variables of structure type.
 *
 * For example:
 * <pre>
 * struct foo { int n; };
 * struct foo f =
 * { // begin initializer-list
 * 1 // end initializer-list
 * };
 * </pre>
 */
public final void initializerList() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST initializerList_AST = null;

    try {      // for error handling
        {
            switch ( LA(1) ) {
            case DOT:
            case LBRACKET:
            {
                designation();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
            }
            break;
        }
        case FLOATING_CONSTANT:
        case CONST_CAST:
        case ENUM:
        case FALSE:
        case REINTERPRET_CAST:
        case SIZEOF:
        case STATIC_CAST:
        case STRUCT:
        case TYPENAME:
        case TRUE:
        case LPAREN:
        case LBRACE:
        case INCR:
        case DECR:
        case PLUS:
        case MINUS:
        case TILDE:
        case BANG:
        case IDENTIFIER:

```

```

case INTEGER_CONSTANT:
case CHAR_CONSTANT:
case STRING_LITERAL:
{
    break;
}
default:
{
    throw new NoViableAltException(LT(1), getFilename());
}
}
}
}
initializer();
if (inputState.guessing==0) {
    astFactory.addASTChild(currentAST, returnAST);
}
}
_loop144:
do {
    if ((LA(1)==COMMA) && (_tokenSet_43.member(LA(2)))) {
        SandboxAST tmp98_AST = null;
        tmp98_AST = (SandboxAST)astFactory.create(LT(1));
        match(COMMA);
        {
            switch ( LA(1) ) {
            case DOT:
            case LBRACKET:
            {
                designation();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
                break;
            }
            case FLOATING_CONSTANT:
            case CONST_CAST:
            case ENUM:
            case FALSE:
            case REINTERPRET_CAST:
            case SIZEOF:
            case STATIC_CAST:
            case STRUCT:
            case TYPENAME:
            case TRUE:
            case LPAREN:
            case LBRACE:
            case INCR:
            case DECR:
            case PLUS:
            case MINUS:
            case TILDE:
            case BANG:
            case IDENTIFIER:
            case INTEGER_CONSTANT:
            case CHAR_CONSTANT:
            case STRING_LITERAL:
            {
                break;
            }
            default:
            {
                throw new NoViableAltException(LT(1), getFilename());
            }
            }
        }
        initializer();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
    }
    else {
        break _loop144;
    }
}

```

```

        } while (true);
    }
    initializerList_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_40);
    } else {
        throw ex;
    }
}
returnAST = initializerList_AST;
}

/**
 * (6.7.8) designation: designated initializers: a new C99 feature
 * that we can probably ignore.
 *
 * A designation can be used when initializing members of a struct
 * in some nice, yet confusing ways. Chances the students will never
 * use them, as <i>I've</i> only just read about them. The April
 * 2001 issue of <u>C++ Users Journal</u> has an article on them,
 * starting on page 56. Here is an example on page 58:
 *
 * struct S1 {
 *     int i;
 *     float f;
 *     int a[2];
 * };
 *
 * struct S1 x = {
 *     .f=3.1,
 *     .i=2,
 *     .a[1]=9
 * };
 *
 * Using this, you can initialize members by name. Interesting, but
 * not likely to be used.
 */
public final void designation() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST designation_AST = null;

    try { // for error handling
        designatorList();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        SandboxAST tmp99_AST = null;
        if (inputState.guessing==0) {
            tmp99_AST = (SandboxAST)astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp99_AST);
        }
        match(ASSIGN);
        designation_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_44);
        } else {
            throw ex;
        }
    }
    returnAST = designation_AST;
}

/** (6.7.8) designator-list: list of designators */
public final void designatorList() throws RecognitionException, TokenStreamException {

```

```

returnAST = null;
ASTPair currentAST = new ASTPair();
SandboxAST designatorList_AST = null;

try {      // for error handling
  {
    int _cnt148=0;
    _loop148:
    do {
      if ((LA(1)==DOT||LA(1)==LBRACKET)) {
        designator();
        if (inputState.guessing==0) {
          astFactory.addASTChild(currentAST, returnAST);
        }
      }
      else {
        if ( _cnt148>=1 ) { break _loop148; } else {throw new NoViableAltException(LT(1), getFilename());}
      }
      _cnt148++;
    } while (true);
  }
  designatorList_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
  if (inputState.guessing==0) {
    reportError(ex);
    consume();
    consumeUntil(_tokenSet_45);
  } else {
    throw ex;
  }
}
returnAST = designatorList_AST;
}

/** (6.7.8) designator: used to initialize array members. */
public final void designator() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
SandboxAST designator_AST = null;

try {      // for error handling
  switch ( LA(1)) {
  case LBRACKET:
  {
    arrayDesignator();
    if (inputState.guessing==0) {
      astFactory.addASTChild(currentAST, returnAST);
    }
  }
  designator_AST = (SandboxAST)currentAST.root;
  break;
}
  case DOT:
  {
    SandboxAST tmp100_AST = null;
    if (inputState.guessing==0) {
      tmp100_AST = (SandboxAST)astFactory.create(LT(1));
      astFactory.addASTChild(currentAST, tmp100_AST);
    }
    match(DOT);
    SandboxAST tmp101_AST = null;
    if (inputState.guessing==0) {
      tmp101_AST = (SandboxAST)astFactory.create(LT(1));
      astFactory.addASTChild(currentAST, tmp101_AST);
    }
    match(IDENTIFIER);
    designator_AST = (SandboxAST)currentAST.root;
    break;
}
  default:
  {
    throw new NoViableAltException(LT(1), getFilename());
  }
}
}

```

```

    }
  }
}
catch (RecognitionException ex) {
  if (inputState.guessing==0) {
    reportError(ex);
    consume();
    consumeUntil(_tokenSet_46);
  } else {
    throw ex;
  }
}
returnAST = designator_AST;
}

/** (6.8) statement: allows interesting things to happen. */
public final void statement() throws RecognitionException, TokenStreamException {

  returnAST = null;
  ASTPair currentAST = new ASTPair();
  SandboxAST statement_AST = null;

  try {      // for error handling
    switch ( LA(1)) {
    case LBRACE:
    {
      compoundStatement();
      if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
      }
      statement_AST = (SandboxAST)currentAST.root;
      break;
    }
    case FLOATING_CONSTANT:
    case CONST_CAST:
    case ENUM:
    case FALSE:
    case REINTERPRET_CAST:
    case SIZEOF:
    case STATIC_CAST:
    case STRUCT:
    case TYPENAME:
    case TRUE:
    case LPAREN:
    case INCR:
    case DECR:
    case PLUS:
    case MINUS:
    case TILDE:
    case BANG:
    case SEMI:
    case IDENTIFIER:
    case INTEGER_CONSTANT:
    case CHAR_CONSTANT:
    case STRING_LITERAL:
    {
      expressionStatement();
      if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
      }
      statement_AST = (SandboxAST)currentAST.root;
      break;
    }
    case IF:
    case SWITCH:
    {
      selectionStatement();
      if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
      }
      statement_AST = (SandboxAST)currentAST.root;
      break;
    }
    case DO:

```

```

    case FOR:
    case WHILE:
    {
        iterationStatement();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        statement_AST = (SandboxAST)currentAST.root;
        break;
    }
    case BREAK:
    case CONTINUE:
    case RETURN:
    {
        jumpStatement();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        statement_AST = (SandboxAST)currentAST.root;
        break;
    }
    default:
    {
        throw new NoViableAltException(LT(1), getFilename());
    }
}
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_47);
    } else {
        throw ex;
    }
}
}
returnAST = statement_AST;
}

/** (6.8.2) compound-statement: a block of statements. */
public final void compoundStatement() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST compoundStatement_AST = null;

    try { // for error handling
        SandboxAST tmp102_AST = null;
        tmp102_AST = (SandboxAST)astFactory.create(LT(1));
        match(LBRACE);
        blockItemList();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        SandboxAST tmp103_AST = null;
        tmp103_AST = (SandboxAST)astFactory.create(LT(1));
        match(RBRACE);
        {
            switch ( LA(1) ) {
            case SL_COMMENT:
            {
                SandboxAST tmp104_AST = null;
                tmp104_AST = (SandboxAST)astFactory.create(LT(1));
                match(SL_COMMENT);
                break;
            }
            case EOF:
            case FLOATING_CONSTANT:
            case BREAK:
            case CASE:
            case CONST:
            case CONST_CAST:
            case CONTINUE:
            case DEFAULT:

```

```

    case DO:
    case ELSE:
    case ENUM:
    case FOR:
    case FALSE:
    case IF:
    case USING:
    case RETURN:
    case REINTERPRET_CAST:
    case SIZEOF:
    case STATIC_CAST:
    case STRUCT:
    case SWITCH:
    case TYPENAME:
    case TYPEDEF:
    case TRUE:
    case WHILE:
    case LPAREN:
    case LBRACE:
    case RBRACE:
    case INCR:
    case DECR:
    case PLUS:
    case MINUS:
    case TILDE:
    case BANG:
    case SEMI:
    case IDENTIFIER:
    case INTEGER_CONSTANT:
    case CHAR_CONSTANT:
    case STRING_LITERAL:
    {
        break;
    }
    default:
    {
        throw new NoViableAltException(LT(1), getFilename());
    }
    }
    if ( inputState.guessing==0 ) {
        compoundStatement_AST = (SandboxAST)currentAST.root;
        compoundStatement_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
            SandboxAST)astFactory.create(BLOCK_STATEMENT,"BLOCK_STATEMENT")).add(compoundStatement_AST));
        currentAST.root = compoundStatement_AST;
        currentAST.child = compoundStatement_AST!=null &&compoundStatement_AST.getFirstChild()!=null ?
            compoundStatement_AST.getFirstChild() : compoundStatement_AST;
        currentAST.advanceChildToEnd();
    }
    compoundStatement_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_48);
    } else {
        throw ex;
    }
}
returnAST = compoundStatement_AST;
}
}

/**
 * (6.8.3) expression-statement: any expression can be a statement.
 *
 * Consider: the expression ‘‘2+2’’ is also the valid statement ‘‘2+2;’’
 */
public final void expressionStatement() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST expressionStatement_AST = null;

```

```

try {      // for error handling
  switch ( LA(1) ) {
  case SEMI:
  {
    SandboxAST tmp105_AST = null;
    tmp105_AST = (SandboxAST)astFactory.create(LT(1));
    match(SEMI);
    expressionStatement_AST = (SandboxAST)currentAST.root;
    break;
  }
  case FLOATING_CONSTANT:
  case CONST_CAST:
  case ENUM:
  case FALSE:
  case REINTERPRET_CAST:
  case SIZEOF:
  case STATIC_CAST:
  case STRUCT:
  case TYPENAME:
  case TRUE:
  case LPAREN:
  case INCR:
  case DECR:
  case PLUS:
  case MINUS:
  case TILDE:
  case BANG:
  case IDENTIFIER:
  case INTEGER_CONSTANT:
  case CHAR_CONSTANT:
  case STRING_LITERAL:
  {
    expression();
    if (inputState.guessing==0) {
      astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp106_AST = null;
    tmp106_AST = (SandboxAST)astFactory.create(LT(1));
    match(SEMI);
    if ( inputState.guessing==0 ) {
      expressionStatement_AST = (SandboxAST)currentAST.root;
      expressionStatement_AST = (SandboxAST)astFactory.make( ( new ASTArray(2)).add((
        SandboxAST)astFactory.create(EXPRESSION_STATEMENT,"EXPRESSION_STATEMENT")).add(expressionStatement_AST));
      currentAST.root = expressionStatement_AST;
      currentAST.child = expressionStatement_AST!=null &&expressionStatement_AST.getFirstChild()!=null ?
        expressionStatement_AST.getFirstChild() : expressionStatement_AST;
      currentAST.advanceChildToEnd();
    }
    expressionStatement_AST = (SandboxAST)currentAST.root;
    break;
  }
  default:
  {
    throw new NoViableAltException(LT(1), getFilename());
  }
  }
}
catch (RecognitionException ex) {
  if (inputState.guessing==0) {
    reportError(ex);
    consume();
    consumeUntil(_tokenSet_47);
  } else {
    throw ex;
  }
}
returnAST = expressionStatement_AST;
}

/** (6.8.4) selection-statement: switch's and if's. */
public final void selectionStatement() throws RecognitionException, TokenStreamException {

  returnAST = null;
  ASTPair currentAST = new ASTPair();

```

```

SandboxAST selectionStatement_AST = null;
Token ifNode = null;
SandboxAST ifNode_AST = null;
Token elseNode = null;
SandboxAST elseNode_AST = null;
Token s = null;
SandboxAST s_AST = null;

try {      // for error handling
  switch ( LA(1) ) {
  case IF:
  {
    ifNode = LT(1);
    if (inputState.guessing==0) {
      ifNode_AST = (SandboxAST)astFactory.create(ifNode);
      astFactory.makeASTRoot(currentAST, ifNode_AST);
    }
    match(IF);
    SandboxAST tmp107_AST = null;
    tmp107_AST = (SandboxAST)astFactory.create(LT(1));
    match(LPAREN);
    expression();
    if (inputState.guessing==0) {
      astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp108_AST = null;
    tmp108_AST = (SandboxAST)astFactory.create(LT(1));
    match(RPAREN);
    statement();
    if (inputState.guessing==0) {
      astFactory.addASTChild(currentAST, returnAST);
    }
    if ( inputState.guessing==0 ) {
      if (!restriction.isEnabled(Restriction.IF))
        reportError ("'if' statements", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
          ifNode.getLine(), ifNode.getColumn());
    }
  }
  {
  if ((LA(1)==ELSE) && (_tokenSet_49.member(LA(2))) && (_tokenSet_50.member(LA(3)))) {
    elseNode = LT(1);
    if (inputState.guessing==0) {
      elseNode_AST = (SandboxAST)astFactory.create(elseNode);
    }
    match(ELSE);
    statement();
    if (inputState.guessing==0) {
      astFactory.addASTChild(currentAST, returnAST);
    }
    if ( inputState.guessing==0 ) {
      if (!restriction.isEnabled(Restriction.IF_ELSE))
        reportError ("'if-then-else' statements", ErrorCodeTable.ARE_DISABLED_2_PARAMS,
          elseNode.getLine(), elseNode.getColumn());
    }
  }
  }
  }
  else if ((_tokenSet_47.member(LA(1))) && (_tokenSet_51.member(LA(2))) && (_tokenSet_52.member(LA(3)))) {
  }
  else {
    throw new NoViableAltException(LT(1), getFilename());
  }
  }
  selectionStatement_AST = (SandboxAST)currentAST.root;
  break;
}
case SWITCH:
{
  s = LT(1);
  if (inputState.guessing==0) {
    s_AST = (SandboxAST)astFactory.create(s);
    astFactory.makeASTRoot(currentAST, s_AST);
  }
  match(SWITCH);
  SandboxAST tmp109_AST = null;
  tmp109_AST = (SandboxAST)astFactory.create(LT(1));

```



```

    if (inputState.guessing==0) {
        w_AST = (SandboxAST)astFactory.create(w);
        astFactory.makeASTRoot(currentAST, w_AST);
    }
    match(WHILE);
    SandboxAST tmp113_AST = null;
    tmp113_AST = (SandboxAST)astFactory.create(LT(1));
    match(LPAREN);
    expression();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp114_AST = null;
    tmp114_AST = (SandboxAST)astFactory.create(LT(1));
    match(RPAREN);
    statement();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    if ( inputState.guessing==0 ) {
        if (!restriction.isEnabled(Restriction.WHILE))
            reportError ("'while' loops", ErrorCodeTable.ARE_DISABLED_2_PARAMS, w.getLine(), w.getColumn());
    }
    iterationStatement_AST = (SandboxAST)currentAST.root;
    break;
}
case DO:
{
    d = LT(1);
    if (inputState.guessing==0) {
        d_AST = (SandboxAST)astFactory.create(d);
        astFactory.makeASTRoot(currentAST, d_AST);
    }
    match(DO);
    statement();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp115_AST = null;
    tmp115_AST = (SandboxAST)astFactory.create(LT(1));
    match(WHILE);
    SandboxAST tmp116_AST = null;
    tmp116_AST = (SandboxAST)astFactory.create(LT(1));
    match(LPAREN);
    expression();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp117_AST = null;
    tmp117_AST = (SandboxAST)astFactory.create(LT(1));
    match(RPAREN);
    SandboxAST tmp118_AST = null;
    tmp118_AST = (SandboxAST)astFactory.create(LT(1));
    match(SEMI);
    if ( inputState.guessing==0 ) {
        if (!restriction.isEnabled(Restriction.DOWHILE))
            reportError ("'do-while' loops", ErrorCodeTable.ARE_DISABLED_2_PARAMS, d.getLine(), d.getColumn());
    }
    iterationStatement_AST = (SandboxAST)currentAST.root;
    break;
}
default:
    boolean synPredMatched180 = false;
    if (((LA(1)==FOR) && (LA(2)==LPAREN) && (_tokenSet_53.member(LA(3)))) {
        int _m180 = mark();
        synPredMatched180 = true;
        inputState.guessing++;
        try {
            {
                match(FOR);
                match(LPAREN);
                declarationSpecifier();
                match(IDENTIFIER);
            }

```

```

    }
    catch (RecognitionException pe) {
        synPredMatched180 = false;
    }
    rewind(_m180);
    inputState.guessing--;
}
if ( synPredMatched180 ) {
    f2 = LT(1);
    if (inputState.guessing==0) {
        f2_AST = (SandboxAST)astFactory.create(f2);
        astFactory.makeASTRoot(currentAST, f2_AST);
    }
    match(FOR);
    SandboxAST tmp119_AST = null;
    tmp119_AST = (SandboxAST)astFactory.create(LT(1));
    match(LPAREN);
    declaration();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    forCond();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp120_AST = null;
    tmp120_AST = (SandboxAST)astFactory.create(LT(1));
    match(SEMI);
    forIter();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp121_AST = null;
    tmp121_AST = (SandboxAST)astFactory.create(LT(1));
    match(RPAREN);
    statement();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    if ( inputState.guessing==0 ) {
        if (!restriction.isEnabled(Restriction.FOR_W_DEC))
            reportError("'for' loops with declarations", ErrorCodeTable.ARE_DISABLED_4_PARAMS,
                f2.getLine(), f2.getColumn());
    }
    iterationStatement_AST = (SandboxAST)currentAST.root;
}
else if ((LA(1)==FOR) && (LA(2)==LPAREN) && (_tokenSet_54.member(LA(3)))) {
    f1 = LT(1);
    if (inputState.guessing==0) {
        f1_AST = (SandboxAST)astFactory.create(f1);
        astFactory.makeASTRoot(currentAST, f1_AST);
    }
    match(FOR);
    SandboxAST tmp122_AST = null;
    tmp122_AST = (SandboxAST)astFactory.create(LT(1));
    match(LPAREN);
    forInit();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp123_AST = null;
    tmp123_AST = (SandboxAST)astFactory.create(LT(1));
    match(SEMI);
    forCond();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    SandboxAST tmp124_AST = null;
    tmp124_AST = (SandboxAST)astFactory.create(LT(1));
    match(SEMI);
    forIter();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
}
}

```



```

    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_47);
        } else {
            throw ex;
        }
    }
    returnAST = jumpStatement_AST;
}

/** (6.8.2) block-item-list: list of statements in a block. */
public final void blockItemList() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST blockItemList_AST = null;

    try {        // for error handling
        {
            _loop157:
            do {
                if ((_tokenSet_55.member(LA(1)))) {
                    blockItem();
                    if (inputState.guessing==0) {
                        astFactory.addASTChild(currentAST, returnAST);
                    }
                }
                else {
                    break _loop157;
                }
            } while (true);
        }
        blockItemList_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_37);
        } else {
            throw ex;
        }
    }
    returnAST = blockItemList_AST;
}

/**
 * (6.8.2) block-item: declarations can be interspersed with statements now,
 * as in C++.
 *
 * Note: This must be kept up in sync with "declaration".
 */
public final void blockItem() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST blockItem_AST = null;

    try {        // for error handling
        boolean synPredMatched160 = false;
        if ((((_tokenSet_53.member(LA(1))) && (_tokenSet_56.member(LA(2))) && (_tokenSet_57.member(LA(3))))) {
            int _m160 = mark();
            synPredMatched160 = true;
            inputState.guessing++;
            try {
                {
                    match(USING);
                }
            }
            catch (RecognitionException pe) {
                synPredMatched160 = false;

```

```

    }
    rewind(_m160);
    inputState.guessing--;
}
if ( synPredMatched160 ) {
    declaration();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    blockItem_AST = (SandboxAST)currentAST.root;
}
else {
    boolean synPredMatched162 = false;
    if (((_tokenSet_53.member(LA(1))) && (_tokenSet_56.member(LA(2))) && (_tokenSet_57.member(LA(3))))) {
        int _m162 = mark();
        synPredMatched162 = true;
        inputState.guessing++;
        try {
            {
                match(TYPEDEF);
            }
        }
        catch (RecognitionException pe) {
            synPredMatched162 = false;
        }
        rewind(_m162);
        inputState.guessing--;
    }
    if ( synPredMatched162 ) {
        declaration();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        blockItem_AST = (SandboxAST)currentAST.root;
    }
    else {
        boolean synPredMatched164 = false;
        if (((_tokenSet_53.member(LA(1))) && (_tokenSet_56.member(LA(2))) && (_tokenSet_57.member(LA(3))))) {
            int _m164 = mark();
            synPredMatched164 = true;
            inputState.guessing++;
            try {
                {
                    declarationSpecifier();
                }
            }
            catch (RecognitionException pe) {
                synPredMatched164 = false;
            }
            rewind(_m164);
            inputState.guessing--;
        }
        if ( synPredMatched164 ) {
            declaration();
            if (inputState.guessing==0) {
                astFactory.addASTChild(currentAST, returnAST);
            }
            blockItem_AST = (SandboxAST)currentAST.root;
        }
        else if (( (_tokenSet_49.member(LA(1))) && (_tokenSet_58.member(LA(2))) && (_tokenSet_59.member(LA(3))))) {
            statement();
            if (inputState.guessing==0) {
                astFactory.addASTChild(currentAST, returnAST);
            }
            blockItem_AST = (SandboxAST)currentAST.root;
        }
        else {
            throw new NoViableAltException(LT(1), getFilename());
        }
    }
}
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
    }
}

```

```

        consume();
        consumeUntil(_tokenSet_60);
    } else {
        throw ex;
    }
}
returnAST = blockItem_AST;
}

/** This was lifted by Peter from the ANTLR example for parsing the Java programming language. */
public final void casesGroup() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST casesGroup_AST = null;

    try {        // for error handling
    {
        int _cnt172=0;
        _loop172:
        do {
            if ((LA(1)==CASE||LA(1)==DEFAULT) && (_tokenSet_61.member(LA(2))) && (_tokenSet_62.member(LA(3)))) {
                acase();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
            }
            else {
                if ( _cnt172>=1 ) { break _loop172; } else {throw new NoViableAltException(LT(1), getFilename());}
            }

            _cnt172++;
        } while (true);
    }
    caseSList();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    if ( inputState.guessing==0 ) {
        casesGroup_AST = (SandboxAST)currentAST.root;
        casesGroup_AST = (SandboxAST)astFactory.make( ( new ASTArray(2)).add((
            SandboxAST)astFactory.create(CASE_GROUP,"CASE_GROUP")).add(casesGroup_AST));
        currentAST.root = casesGroup_AST;
        currentAST.child = casesGroup_AST!=null &&casesGroup_AST.getFirstChild()!=null ?
            casesGroup_AST.getFirstChild() : casesGroup_AST;
        currentAST.advanceChildToEnd();
    }
    casesGroup_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_63);
    } else {
        throw ex;
    }
}
returnAST = casesGroup_AST;
}

public final void acase() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST acase_AST = null;

    try {        // for error handling
    {
        switch ( LA(1) ) {
        case CASE:
        {
            SandboxAST tmp129_AST = null;
            if (inputState.guessing==0) {

```

```

        tmp129_AST = (SandboxAST)astFactory.create(LT(1));
        astFactory.makeASTRoot(currentAST, tmp129_AST);
    }
    match(CASE);
    expression();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    break;
}
case DEFAULT:
{
    SandboxAST tmp130_AST = null;
    if (inputState.guessing==0) {
        tmp130_AST = (SandboxAST)astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp130_AST);
    }
    match(DEFAULT);
    break;
}
default:
{
    throw new NoViableAltException(LT(1), getFilename());
}
}
SandboxAST tmp131_AST = null;
tmp131_AST = (SandboxAST)astFactory.create(LT(1));
match(COLON);
acase_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_64);
    }
    else {
        throw ex;
    }
}
returnAST = acase_AST;
}
}

public final void caseSList() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST caseSList_AST = null;

    try { // for error handling
    {
        _loop177:
        do {
            if ((_tokenSet_49.member(LA(1)))) {
                statement();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
            }
            else {
                break _loop177;
            }
        } while (true);
    }
    if ( inputState.guessing==0 ) {
        caseSList_AST = (SandboxAST)currentAST.root;
        caseSList_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
            SandboxAST)astFactory.create(SLIST,"SLIST")).add(caseSList_AST));
        currentAST.root = caseSList_AST;
        currentAST.child = caseSList_AST!=null &&caseSList_AST.getFirstChild()!=null ?
            caseSList_AST.getFirstChild() : caseSList_AST;
        currentAST.advanceChildToEnd();
    }
}
}

```

```

        caseSList_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_63);
        } else {
            throw ex;
        }
    }
}
returnAST = caseSList_AST;
}

public final void forCond() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST forCond_AST = null;

    try {        // for error handling
        expression();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        if ( inputState.guessing==0 ) {
            forCond_AST = (SandboxAST)currentAST.root;
            forCond_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
                SandboxAST)astFactory.create(FOR_COND,"FOR_COND")).add(forCond_AST));
            currentAST.root = forCond_AST;
            currentAST.child = forCond_AST!=null &&forCond_AST.getFirstChild()!=null ?
                forCond_AST.getFirstChild() : forCond_AST;
            currentAST.advanceChildToEnd();
        }
        forCond_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_31);
        } else {
            throw ex;
        }
    }
}
returnAST = forCond_AST;
}

public final void forIter() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST forIter_AST = null;

    try {        // for error handling
    {
        switch ( LA(1) ) {
        case FLOATING_CONSTANT:
        case CONST_CAST:
        case ENUM:
        case FALSE:
        case REINTERPRET_CAST:
        case SIZEOF:
        case STATIC_CAST:
        case STRUCT:
        case TYPENAME:
        case TRUE:
        case LPAREN:
        case INCR:
        case DECR:
        case PLUS:
        case MINUS:
        case TILDE:
        case BANG:

```

```

case IDENTIFIER:
case INTEGER_CONSTANT:
case CHAR_CONSTANT:
case STRING_LITERAL:
{
    expression();
    if (inputState.guessing==0) {
        astFactory.addASTChild(currentAST, returnAST);
    }
    break;
}
case RPAREN:
{
    break;
}
default:
{
    throw new NoViableAltException(LT(1), getFilename());
}
}
}
}
if ( inputState.guessing==0 ) {
    forIter_AST = (SandboxAST)currentAST.root;
    forIter_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
        SandboxAST)astFactory.create(FOR_ITER,"FOR_ITER")).add(forIter_AST));
    currentAST.root = forIter_AST;
    currentAST.child = forIter_AST!=null &&forIter_AST.getFirstChild()!=null ?
        forIter_AST.getFirstChild() : forIter_AST;
    currentAST.advanceChildToEnd();
}
forIter_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_6);
    } else {
        throw ex;
    }
}
returnAST = forIter_AST;
}

public final void forInit() throws RecognitionException, TokenStreamException {

returnAST = null;
ASTPair currentAST = new ASTPair();
SandboxAST forInit_AST = null;

try {    // for error handling
{
    switch ( LA(1) ) {
    case FLOATING_CONSTANT:
    case CONST_CAST:
    case ENUM:
    case FALSE:
    case REINTERPRET_CAST:
    case SIZEOF:
    case STATIC_CAST:
    case STRUCT:
    case TYPENAME:
    case TRUE:
    case LPAREN:
    case INCR:
    case DECR:
    case PLUS:
    case MINUS:
    case TILDE:
    case BANG:
    case IDENTIFIER:
    case INTEGER_CONSTANT:
    case CHAR_CONSTANT:
    case STRING_LITERAL:

```

```

    {
        expression();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        break;
    }
    case SEMI:
    {
        break;
    }
    default:
    {
        throw new NoViableAltException(LT(1), getFilename());
    }
}
}
if ( inputState.guessing==0 ) {
    forInit_AST = (SandboxAST)currentAST.root;
    forInit_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
        SandboxAST)astFactory.create(FOR_INIT,"FOR_INIT")).add(forInit_AST));
    currentAST.root = forInit_AST;
    currentAST.child = forInit_AST!=null &&forInit_AST.getFirstChild()!=null ?
        forInit_AST.getFirstChild() : forInit_AST;
    currentAST.advanceChildToEnd();
}
forInit_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_31);
    } else {
        throw ex;
    }
}
}
returnAST = forInit_AST;
}

/** (6.10) preprocessing-file: a single source file. */
public final void preprocessingFile() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST preprocessingFile_AST = null;

    try { // for error handling
        group();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        preprocessingFile_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_65);
        } else {
            throw ex;
        }
    }
}
returnAST = preprocessingFile_AST;
}

/**
 * (6.9) external-declaration: declarations for types that are not defined
 * (storage-allocated) in this file.
 */
public final void externalDeclaration() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();

```

```

SandboxAST externalDeclaration_AST = null;

try {      // for error handling
    boolean synPredMatched197 = false;
    if (((_tokenSet_29.member(LA(1))) && (_tokenSet_66.member(LA(2))) && (_tokenSet_67.member(LA(3))))) {
        int _m197 = mark();
        synPredMatched197 = true;
        inputState.guessing++;
        try {
            {
                {
                    int _cnt196=0;
                    _loop196:
                    do {
                        if ((_tokenSet_29.member(LA(1)))) {
                            declarationSpecifier();
                        }
                        else {
                            if ( _cnt196>=1 ) { break _loop196; } else {throw new NoViableAltException(LT(1), getFilename());}
                        }

                        _cnt196++;
                    } while (true);
                }
                match(IDENTIFIER);
                match(LPAREN);
            }
        }
        catch (RecognitionException pe) {
            synPredMatched197 = false;
        }
        rewind(_m197);
        inputState.guessing--;
    }
    if ( synPredMatched197 ) {
        functionDefinition();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        externalDeclaration_AST = (SandboxAST)currentAST.root;
    }
    else if ((_tokenSet_53.member(LA(1))) && (_tokenSet_56.member(LA(2))) && (_tokenSet_68.member(LA(3))))) {
        declaration();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        externalDeclaration_AST = (SandboxAST)currentAST.root;
    }
    else if ((LA(1)==SEMI)) {
        SandboxAST tmp132_AST = null;
        tmp132_AST = (SandboxAST)astFactory.create(LT(1));
        match(SEMI);
        externalDeclaration_AST = (SandboxAST)currentAST.root;
    }
    else {
        throw new NoViableAltException(LT(1), getFilename());
    }
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_69);
    }
    else {
        throw ex;
    }
}
returnAST = externalDeclaration_AST;
}

/**
 * (6.9.1) function-definition: the implementation of a function.
 *

```

```

* The grammar appears to allow for K&R-style variable declarations,
* which I had though were (finally) removed. We're not supporting
* K&R style, only ANSI-style.
*
* This isn't a direct equivalent of the standard, but it's sufficient.
*/
public final void functionDefinition() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST functionDefinition_AST = null;

    try {        // for error handling
    {
        int _cnt200=0;
        _loop200:
        do {
            if ((_tokenSet_29.member(LA(1)))) {
                declarationSpecifier();
                if (inputState.guessing==0) {
                    astFactory.addASTChild(currentAST, returnAST);
                }
            }
            else {
                if ( _cnt200>=1 ) { break _loop200; } else {throw new NoViableAltException(LT(1), getFilename());}
            }
        }
        _cnt200++;
    } while (true);
    }
    SandboxAST tmp133_AST = null;
    if (inputState.guessing==0) {
        tmp133_AST = (SandboxAST)astFactory.create(LT(1));
        astFactory.addASTChild(currentAST, tmp133_AST);
    }
    match(IDENTIFIER);
    SandboxAST tmp134_AST = null;
    tmp134_AST = (SandboxAST)astFactory.create(LT(1));
    match(LPAREN);
    {
        switch ( LA(1) ) {
        case CONST:
        case ENUM:
        case STRUCT:
        case TYPENAME:
        {
            parameterList();
            if (inputState.guessing==0) {
                astFactory.addASTChild(currentAST, returnAST);
            }
        }
        break;
        case RPAREN:
        {
            break;
        }
        default:
        {
            throw new NoViableAltException(LT(1), getFilename());
        }
    }
    }
    SandboxAST tmp135_AST = null;
    tmp135_AST = (SandboxAST)astFactory.create(LT(1));
    match(RPAREN);
    {
        switch ( LA(1) ) {
        case LBRACE:
        {
            compoundStatement();
            if (inputState.guessing==0) {
                astFactory.addASTChild(currentAST, returnAST);
            }
        }
        if ( inputState.guessing==0 ) {

```

```

        functionDefinition_AST = (SandboxAST)currentAST.root;
        functionDefinition_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
            SandboxAST)astFactory.create(FUNCTION,"FUNCTION")).add(functionDefinition_AST));
        currentAST.root = functionDefinition_AST;
        currentAST.child = functionDefinition_AST!=null &&functionDefinition_AST.getFirstChild()!=null ?
            functionDefinition_AST.getFirstChild() : functionDefinition_AST;
        currentAST.advanceChildToEnd();
    }
    break;
}
case SEMI:
{
    SandboxAST tmp136_AST = null;
    tmp136_AST = (SandboxAST)astFactory.create(LT(1));
    match(SEMI);
    if ( inputState.guessing==0 ) {
        functionDefinition_AST = (SandboxAST)currentAST.root;
        functionDefinition_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
            SandboxAST)astFactory.create(FUNCTION_PROTOTYPE,"FUNCTION_PROTOTYPE")).add(functionDefinition_AST));
        currentAST.root = functionDefinition_AST;
        currentAST.child = functionDefinition_AST!=null &&functionDefinition_AST.getFirstChild()!=null ?
            functionDefinition_AST.getFirstChild() : functionDefinition_AST;
        currentAST.advanceChildToEnd();
    }
    break;
}
default:
{
    throw new NoViableAltException(LT(1), getFilename());
}
}
}
functionDefinition_AST = (SandboxAST)currentAST.root;
}
catch (RecognitionException ex) {
    if (inputState.guessing==0) {
        reportError(ex);
        consume();
        consumeUntil(_tokenSet_69);
    } else {
        throw ex;
    }
}
}
returnAST = functionDefinition_AST;
}
}

/** (6.10) group: a group of preprocessor statements. */
public final void group() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST group_AST = null;

    try {        // for error handling
        groupPart();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        group_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_65);
        } else {
            throw ex;
        }
    }
    returnAST = group_AST;
}
}

/**
 * (6.10) group-part: #if blocks, control lines, normal text, etc.

```

```

*
* We only care about '#include' statements, so we're doing
* a small subset.
*/
public final void groupPart() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST groupPart_AST = null;

    try {        // for error handling
        controllLine();
        if (inputState.guessing==0) {
            astFactory.addASTChild(currentAST, returnAST);
        }
        groupPart_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_65);
        } else {
            throw ex;
        }
    }
    returnAST = groupPart_AST;
}

/**
* (6.10) control-line: "commands" for the preprocessor, which include
* '#include' statements, '#define' statements, macros, and other
* preprocessor commands not dealing with '#if'.
*/
public final void controllLine() throws RecognitionException, TokenStreamException {

    returnAST = null;
    ASTPair currentAST = new ASTPair();
    SandboxAST controllLine_AST = null;

    try {        // for error handling
        SandboxAST tmp137_AST = null;
        if (inputState.guessing==0) {
            tmp137_AST = (SandboxAST)astFactory.create(LT(1));
            astFactory.addASTChild(currentAST, tmp137_AST);
        }
        match(POUND_INCLUDE_DIRECTIVE);
        if ( inputState.guessing==0 ) {
            controllLine_AST = (SandboxAST)currentAST.root;
            controllLine_AST = (SandboxAST)astFactory.make( (new ASTArray(2)).add((
                SandboxAST)astFactory.create(INCLUDE,"INCLUDE")).add(controllLine_AST));
            currentAST.root = controllLine_AST;
            currentAST.child = controllLine_AST!=null &&controllLine_AST.getFirstChild()!=null ?
                controllLine_AST.getFirstChild() : controllLine_AST;
            currentAST.advanceChildToEnd();
        }
        controllLine_AST = (SandboxAST)currentAST.root;
    }
    catch (RecognitionException ex) {
        if (inputState.guessing==0) {
            reportError(ex);
            consume();
            consumeUntil(_tokenSet_65);
        } else {
            throw ex;
        }
    }
    returnAST = controllLine_AST;
}

public static final String[] _tokenNames = {
    "<0>",
    "EOF",
    "<2>",

```

```
"NULL_TREE_LOOKAHEAD",
"DOT",
"FLOATING_CONSTANT",
"break",
"case",
"const",
"const_cast",
"continue",
"default",
"do",
"else",
"enum",
"for",
>false",
"if",
"namespace",
"using",
"return",
"reinterpret_cast",
"sizeof",
"static_cast",
"struct",
"switch",
"TYPENAME",
"typedef",
>true",
"while",
"LBACKET",
"RBACKET",
"LPAREN",
"RPAREN",
"LBACE",
"RBRACE",
"INCR",
"DECR",
"AMPERSAND",
"STAR",
"PLUS",
"MINUS",
"TILDE",
"BANG",
"SLASH",
"PERCENT",
"LSHIFT",
"RSHIFT",
"LT",
"GT",
"LE",
"GE",
"EQ",
"NE",
"CARET",
"PIPE",
"LAND",
"LOR",
"QUESTION",
"COLON",
"COLON_COLON",
"SEMI",
"ASSIGN",
"STAR_ASSIGN",
"SLASH_ASSIGN",
"PERCENT_ASSIGN",
"PLUS_ASSIGN",
"MINUS_ASSIGN",
"COMMA",
"POUND",
"LSHIFT_ASSIGN",
"RSHIFT_ASSIGN",
"AMPERSAND_ASSIGN",
"CARET_ASSIGN",
"PIPE_ASSIGN",
"WS",
"WHITESPACE",
```

```

"NEW_LINE",
"SL_COMMENT",
"ML_COMMENT",
"IDENTIFIER",
"IDENTIFIER_NONDIGIT",
"DIGIT",
"INTEGER_CONSTANT",
"EXPONENT",
"FLOATING_SUFFIX",
"INTEGER_SUFFIX",
"CHAR_CONSTANT",
"STRING_LITERAL",
"ESC",
"HEADER_NAME",
"PP_TOKENS",
"PREPROCESSING_TOKEN",
"POUND_INCLUDE_DIRECTIVE",
"ARGUMENTS",
"ARRAY_DECLARATOR",
"BLOCK_STATEMENT",
"CALL",
"CASE_GROUP",
"CAST",
"DECLARATION",
"DECLARATOR",
"ENUMERATOR_LIST",
"EXPRESSION_STATEMENT",
"FOR_INIT",
"FOR_COND",
"FOR_ITER",
"FUNCTION",
"FUNCTION_PROTOTYPE",
"INCLUDE",
"INITIALIZER",
"PARAMETER",
"PARAMETER_LIST",
"POST_DECR",
"POST_INCR",
"PROGRAM",
"SLIST",
"STRUCT_DECLARATION",
"STRUCT_MEMBERS",
"SUBSCRIPT",
"TYPE_MODIFIER",
"TYPE_SIZEOF",
"USING_DIRECTIVE"
};

private static final long _tokenSet_0_data_[] = { 2L, 0L };
public static final BitSet _tokenSet_0 = new BitSet(_tokenSet_0_data_);
private static final long _tokenSet_1_data_[] = { 2305843009432338688L, 0L };
public static final BitSet _tokenSet_1 = new BitSet(_tokenSet_1_data_);
private static final long _tokenSet_2_data_[] = { -342287058951208944L, 31L, 0L, 0L };
public static final BitSet _tokenSet_2 = new BitSet(_tokenSet_2_data_);
private static final long _tokenSet_3_data_[] = { 2882303772254535680L, 0L };
public static final BitSet _tokenSet_3 = new BitSet(_tokenSet_3_data_);
private static final long _tokenSet_4_data_[] = { -1495208774011453440L, 31L, 0L, 0L };
public static final BitSet _tokenSet_4 = new BitSet(_tokenSet_4_data_);
private static final long _tokenSet_5_data_[] = { 2306406247013826818L, 65552L, 0L, 0L };
public static final BitSet _tokenSet_5 = new BitSet(_tokenSet_5_data_);
private static final long _tokenSet_6_data_[] = { 8589934592L, 0L };
public static final BitSet _tokenSet_6 = new BitSet(_tokenSet_6_data_);
private static final long _tokenSet_7_data_[] = { 4563468320L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_7 = new BitSet(_tokenSet_7_data_);
private static final long _tokenSet_8_data_[] = { 16703494898224L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_8 = new BitSet(_tokenSet_8_data_);
private static final long _tokenSet_9_data_[] = { -3224577645289061840L, 25755679L, 0L, 0L };
public static final BitSet _tokenSet_9 = new BitSet(_tokenSet_9_data_);
private static final long _tokenSet_10_data_[] = { 4657857056L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_10 = new BitSet(_tokenSet_10_data_);
private static final long _tokenSet_11_data_[] = { -342273847264722384L, 25755679L, 0L, 0L };
public static final BitSet _tokenSet_11 = new BitSet(_tokenSet_11_data_);
private static final long _tokenSet_12_data_[] = { -342273846558326798L, 25755679L, 0L, 0L };
public static final BitSet _tokenSet_12 = new BitSet(_tokenSet_12_data_);

```

```

private static final long _tokenSet_13_data_[] = { -342273848338464208L, 25755679L, 0L, 0L };
public static final BitSet _tokenSet_13 = new BitSet(_tokenSet_13_data_);
private static final long _tokenSet_14_data_[] = { -1495208775085195264L, 31L, 0L, 0L };
public static final BitSet _tokenSet_14 = new BitSet(_tokenSet_14_data_);
private static final long _tokenSet_15_data_[] = { 16703490703904L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_15 = new BitSet(_tokenSet_15_data_);
private static final long _tokenSet_16_data_[] = { -3458466317095321040L, 25755663L, 0L, 0L };
public static final BitSet _tokenSet_16 = new BitSet(_tokenSet_16_data_);
private static final long _tokenSet_17_data_[] = { -3224577610929323216L, 25755679L, 0L, 0L };
public static final BitSet _tokenSet_17 = new BitSet(_tokenSet_17_data_);
private static final long _tokenSet_18_data_[] = { 16703494898208L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_18 = new BitSet(_tokenSet_18_data_);
private static final long _tokenSet_19_data_[] = { 4269412171162665520L, 25755664L, 0L, 0L };
public static final BitSet _tokenSet_19 = new BitSet(_tokenSet_19_data_);
private static final long _tokenSet_20_data_[] = { 2882303806614274048L, 16L, 0L, 0L };
public static final BitSet _tokenSet_20 = new BitSet(_tokenSet_20_data_);
private static final long _tokenSet_21_data_[] = { 25836912640L, 65536L, 0L, 0L };
public static final BitSet _tokenSet_21 = new BitSet(_tokenSet_21_data_);
private static final long _tokenSet_22_data_[] = { 3116423917028245504L, 16L, 0L, 0L };
public static final BitSet _tokenSet_22 = new BitSet(_tokenSet_22_data_);
private static final long _tokenSet_23_data_[] = { 3116420618493362176L, 16L, 0L, 0L };
public static final BitSet _tokenSet_23 = new BitSet(_tokenSet_23_data_);
private static final long _tokenSet_24_data_[] = { 3116209512260829184L, 16L, 0L, 0L };
public static final BitSet _tokenSet_24 = new BitSet(_tokenSet_24_data_);
private static final long _tokenSet_25_data_[] = { 3111987387610169344L, 16L, 0L, 0L };
public static final BitSet _tokenSet_25 = new BitSet(_tokenSet_25_data_);
private static final long _tokenSet_26_data_[] = { 3098476588728057856L, 16L, 0L, 0L };
public static final BitSet _tokenSet_26 = new BitSet(_tokenSet_26_data_);
private static final long _tokenSet_27_data_[] = { 3026418994690129920L, 16L, 0L, 0L };
public static final BitSet _tokenSet_27 = new BitSet(_tokenSet_27_data_);
private static final long _tokenSet_28_data_[] = { 2147483648L, 0L };
public static final BitSet _tokenSet_28 = new BitSet(_tokenSet_28_data_);
private static final long _tokenSet_29_data_[] = { 83902720L, 0L };
public static final BitSet _tokenSet_29 = new BitSet(_tokenSet_29_data_);
private static final long _tokenSet_30_data_[] = { 2305859764954584930L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_30 = new BitSet(_tokenSet_30_data_);
private static final long _tokenSet_31_data_[] = { 2305843009213693952L, 0L };
public static final BitSet _tokenSet_31 = new BitSet(_tokenSet_31_data_);
private static final long _tokenSet_32_data_[] = { 2305843017887531264L, 65552L, 0L, 0L };
public static final BitSet _tokenSet_32 = new BitSet(_tokenSet_32_data_);
private static final long _tokenSet_33_data_[] = { 83902464L, 0L };
public static final BitSet _tokenSet_33 = new BitSet(_tokenSet_33_data_);
private static final long _tokenSet_34_data_[] = { 2305843009213693952L, 16L, 0L, 0L };
public static final BitSet _tokenSet_34 = new BitSet(_tokenSet_34_data_);
private static final long _tokenSet_35_data_[] = { 6917529036231016448L, 16L, 0L, 0L };
public static final BitSet _tokenSet_35 = new BitSet(_tokenSet_35_data_);
private static final long _tokenSet_36_data_[] = { 2305843043573432320L, 16L, 0L, 0L };
public static final BitSet _tokenSet_36 = new BitSet(_tokenSet_36_data_);
private static final long _tokenSet_37_data_[] = { 34359738368L, 0L };
public static final BitSet _tokenSet_37 = new BitSet(_tokenSet_37_data_);
private static final long _tokenSet_38_data_[] = { 34443641088L, 0L };
public static final BitSet _tokenSet_38 = new BitSet(_tokenSet_38_data_);
private static final long _tokenSet_39_data_[] = { 2305843009213693954L, 65536L, 0L, 0L };
public static final BitSet _tokenSet_39 = new BitSet(_tokenSet_39_data_);
private static final long _tokenSet_40_data_[] = { 34359738368L, 16L, 0L, 0L };
public static final BitSet _tokenSet_40 = new BitSet(_tokenSet_40_data_);
private static final long _tokenSet_41_data_[] = { 6917529037304758288L, 16L, 0L, 0L };
public static final BitSet _tokenSet_41 = new BitSet(_tokenSet_41_data_);
private static final long _tokenSet_42_data_[] = { 8589934592L, 16L, 0L, 0L };
public static final BitSet _tokenSet_42 = new BitSet(_tokenSet_42_data_);
private static final long _tokenSet_43_data_[] = { 16721748509232L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_43 = new BitSet(_tokenSet_43_data_);
private static final long _tokenSet_44_data_[] = { 16720674767392L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_44 = new BitSet(_tokenSet_44_data_);
private static final long _tokenSet_45_data_[] = { 4611686018427387904L, 0L };
public static final BitSet _tokenSet_45 = new BitSet(_tokenSet_45_data_);
private static final long _tokenSet_46_data_[] = { 4611686019501129744L, 0L };
public static final BitSet _tokenSet_46 = new BitSet(_tokenSet_46_data_);
private static final long _tokenSet_47_data_[] = { 2305859764954595296L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_47 = new BitSet(_tokenSet_47_data_);
private static final long _tokenSet_48_data_[] = { 2305859764954595298L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_48 = new BitSet(_tokenSet_48_data_);
private static final long _tokenSet_49_data_[] = { 2305859730460104288L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_49 = new BitSet(_tokenSet_49_data_);

```

```
private static final long _tokenSet_50_data_[] = { -918734609599168528L, 25755679L, 0L, 0L };
public static final BitSet _tokenSet_50 = new BitSet(_tokenSet_50_data_);
private static final long _tokenSet_51_data_[] = { -342273582417575950L, 25772063L, 0L, 0L };
public static final BitSet _tokenSet_51 = new BitSet(_tokenSet_51_data_);
private static final long _tokenSet_52_data_[] = { -342273573827641358L, 25772063L, 0L, 0L };
public static final BitSet _tokenSet_52 = new BitSet(_tokenSet_52_data_);
private static final long _tokenSet_53_data_[] = { 218644736L, 0L };
public static final BitSet _tokenSet_53 = new BitSet(_tokenSet_53_data_);
private static final long _tokenSet_54_data_[] = { 2305859712708592160L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_54 = new BitSet(_tokenSet_54_data_);
private static final long _tokenSet_55_data_[] = { 2305859730594846560L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_55 = new BitSet(_tokenSet_55_data_);
private static final long _tokenSet_56_data_[] = { 2305843301355634944L, 65536L, 0L, 0L };
public static final BitSet _tokenSet_56 = new BitSet(_tokenSet_56_data_);
private static final long _tokenSet_57_data_[] = { 6917546059333621600L, 25755664L, 0L, 0L };
public static final BitSet _tokenSet_57 = new BitSet(_tokenSet_57_data_);
private static final long _tokenSet_58_data_[] = { -918734609599178896L, 25755679L, 0L, 0L };
public static final BitSet _tokenSet_58 = new BitSet(_tokenSet_58_data_);
private static final long _tokenSet_59_data_[] = { -918734326131064846L, 25772063L, 0L, 0L };
public static final BitSet _tokenSet_59 = new BitSet(_tokenSet_59_data_);
private static final long _tokenSet_60_data_[] = { 2305859764954584928L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_60 = new BitSet(_tokenSet_60_data_);
private static final long _tokenSet_61_data_[] = { 576477455798321696L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_61 = new BitSet(_tokenSet_61_data_);
private static final long _tokenSet_62_data_[] = { -342273857430495504L, 25755679L, 0L, 0L };
public static final BitSet _tokenSet_62 = new BitSet(_tokenSet_62_data_);
private static final long _tokenSet_63_data_[] = { 34359740544L, 0L };
public static final BitSet _tokenSet_63 = new BitSet(_tokenSet_63_data_);
private static final long _tokenSet_64_data_[] = { 2305859764819844832L, 25755648L, 0L, 0L };
public static final BitSet _tokenSet_64 = new BitSet(_tokenSet_64_data_);
private static final long _tokenSet_65_data_[] = { 2305843009432338690L, 536870912L, 0L, 0L };
public static final BitSet _tokenSet_65 = new BitSet(_tokenSet_65_data_);
private static final long _tokenSet_66_data_[] = { 292141678848L, 65536L, 0L, 0L };
public static final BitSet _tokenSet_66 = new BitSet(_tokenSet_66_data_);
private static final long _tokenSet_67_data_[] = { 330796384512L, 65536L, 0L, 0L };
public static final BitSet _tokenSet_67 = new BitSet(_tokenSet_67_data_);
private static final long _tokenSet_68_data_[] = { 6917529359645950210L, 65552L, 0L, 0L };
public static final BitSet _tokenSet_68 = new BitSet(_tokenSet_68_data_);
private static final long _tokenSet_69_data_[] = { 2305843009432338690L, 0L };
public static final BitSet _tokenSet_69 = new BitSet(_tokenSet_69_data_);
}
```

J.5 ANTLR Input File Used to Create *CS1 Sandbox*'s Semantic Validator (Semantic.g)

```

header {
package sandbox.compiler.backend;

import java.io.*;
import java.lang.*;
import java.lang.reflect.*;
import java.util.*;
import sandbox.lib.*;
import sandbox.lib.iostream.*;
import sandbox.client.*;
import sandbox.compiler.*;
import sandbox.compiler.type.*;
import sandbox.compiler.syntab.*;
import sandbox.compiler.frontend.*;
import antlr.*;
}

/**
 *
 * Performs the semantic check on the parsed AST.
 */

class NewSemantic extends TreeParser;
options {
    importVocab = CLexer;
    importVocab = CParser;
}

{
    // Boolean indicator for the presense of the 'using' declaration -- used to enforce
    // 'using namespace std' for I/O statements (like cout!)
    private boolean usingStdSet = false;
    private boolean usingErrMsgShown = false;

    // Boolean used to validate function returns are present, when needed
    private boolean returnSeen = false;

    // The symbol table
    SymTab syntab = new LinkSymTab();

    // The errors that are observed during the parse phase
    private Vector errorVector = new Vector();
    private Vector typesDeclared = new Vector();

    // The library manager (which contains the type manager)
    private LibraryManager libraryManager = null;
    private TypeManager typeManager = null;

    // The C language features that are allowed/forbidden.
    private Restriction restriction;

    // Output streams for debugging
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    PrintStream debugStream = new PrintStream(out);

    // Vector of cases, used in switchStatement and labeledStatement
    Stack switchLabelStack = new Stack();

    // Retains a reference to the wrapper of the current function being examined.
    // This is so we can obtain a reference to its AST when we check the return type.
    private FunctionWrapper currentFunction = null;

    // The function manager, manages the addition, use and query of user-defined functions
    private FunctionManager fm;

/**
 * Sets the library manager to the parameter, also sets the

```

```
* reference to the type manager.
*/
public void setLibraryManager(LibraryManager mgr) {
    libraryManager = mgr;
    typeManager = libraryManager.getTypeManager();
}

/**
 * Constructor which accepts a reference to the function manager.
 */
public NewSemantic (FunctionManager funcMgr) {
    fm = funcMgr;
}

/**
 * Sets a reference to the current restriction object
 */
public void setRestriction(Restriction restr) {
    restriction = restr;
}

/**
 * Returns a vector of each error seen during semantic checking
 */
public Vector getErrorVector() {
    return(errorVector);
}

/**
 * Overridden reportError method, supporting the error reporting via ANTLR
 */
public void reportError (int code, int line, int col) {
    reportError("", code, line, col);
}

/**
 * ReportError method, supporting the error reporting similiar to what ANTLR does.
 * We added support for the line number and column via this method.
 */
public void reportError (java.lang.String errMsg, int code, int line, int col) {
    AntlrError e = new AntlrError ();
    e.setType("semantic check error");
    e.setErrorMessage(errMsg);
    e.setErrorCode(code);
    e.setLineNumber(line);
    e.setColumnNumber(col);
    errorVector.add(e);
}

/**
 * Overridden reportError method, supporting the error reporting via ANTLR
 */
public void reportError(RecognitionException ex) {
    AntlrError error = new AntlrError();
    System.err.println("Recognition Exception in semantic checker: " + ex.getMessage() + ", line:" + ex.getLine());
    error.setType("recognition exception - semantic checker");
    error.setErrorMessage(ex.getMessage());
    error.setLineNumber(ex.getLine());
    error.setColumnNumber(ex.getColumn());
    errorVector.add(error);
    ex.printStackTrace();
}

/**
 * Overridden reportError method, supporting the error reporting via ANTLR.
 * There is no support for the line number or column number via this method.
 */
```

```

public void reportError(java.lang.String s) {
    AntlrError error = new AntlrError();
    error.setType("error - semantic checker");
    error.setErrorMessage(s);
    errorVector.add(error);
}

/**
 * Overridden reportError method, supporting the error reporting via ANTLR
 */
public void reportWarning(java.lang.String s) {
    AntlrError error = new AntlrError();
    error.setType("warning");
    error.setErrorMessage(s);
    errorVector.add(error);
}

/**
 * Sets the debugging stream
 */
public void setDebugOutput(PrintStream p) {
    debugStream = p;
}

/**
 * Returns the string representation of the debugging output, for display in
 * debugging window.
 */
public java.lang.String getDebugOutput() {
    return(out.toString());
}

/**
 * Returns the String representation of the identifier 'identifName's' type, assuming it exists.
 */
private String getType(String identifName) {
    String returnType = null;

    if ((symtab != null) && (identifName != null)) {
        AbstractRepresentation abRep = symtab.getRepresentation(identifName);
        if (abRep != null)
            returnType = abRep.getType().getName();
    }
    return returnType;
}

/**
 * Validates the assignment of the specified types. Line number and
 * column number are included for use in the error output (if needed). Returns the
 * left hand type's type as the resulting type of the expression.
 */
AbstractType checkAssignment(int op, AbstractType lType, AbstractType rType, int line, int col) {
    boolean valid = false;

    if (rType != null && lType != null) {
        debugStream.println("    Checking result of assignment of " + rType.getName() +
            " to " + lType.getName());

        try {
            valid = typeManager.checkConversion(lType, rType);
        } catch (IncompatibleTypesException ex) {
            valid = false;
        }

        if (!valid && rType != null && lType != null) {
            reportError(lType.getName() + " " + rType.getName(), ErrorCodeTable.INCOMPATIBLE_TYPES, line, col);
            debugStream.println("    " + lType.getName() + " and " + rType.getName() +
                " are incompatible data types.");
        }
    }
}

```

```

        debugStream.println("    Assignment results in " + lType.getName() + " type");
    } else {
        debugStream.print("    Insufficient type information: (");
        if (lType != null)
            debugStream.print("LHS: " + lType);
        else
            debugStream.print("LHS: null");

        if (rType != null)
            debugStream.print("    RHS: " + rType);
        else
            debugStream.print("    RHS: null");
        debugStream.println(")");
    }
    return lType;
}

/**
 * Validates the use of the binary operation on the specified types. Line number and
 * column number are included for use in the error output (if needed). Returns the
 * type of value returned from the expression.
 */
AbstractType checkBinaryOp(int op, AbstractType lType, AbstractType rType, int line, int col) {
    AbstractType type = null;

    if (rType != null && lType != null)
        debugStream.println("    Checking result of operation between " + lType.getName() + " and " +
            rType.getName());
    else {
        debugStream.print("    Insufficient type information: (");
        if (lType != null)
            debugStream.print("LHS: " + lType);
        else
            debugStream.print("LHS: null");

        if (rType != null)
            debugStream.print("    RHS: " + rType);
        else
            debugStream.print("    RHS: null");
        debugStream.println(")");
    }

    try {
        type = typeManager.checkBinaryOp(op, lType, rType);
    } catch (IncompatibleTypesException ite) {
        reportError(lType.getName() + " " + rType.getName(), ErrorCodeTable.INCOMPATIBLE_TYPES, line, col);
        debugStream.println("    " + lType.getName() + " and " + rType.getName() +
            " are incompatible data types.");
    } catch (InvalidOperationException ioe) {
        reportError(lType.getName() + " " + rType.getName(), ErrorCodeTable.INVALID_OPERATION, line, col);
        debugStream.println("    Operation between " + lType.getName() + " and " + rType.getName() +
            " not supported");
    }
}
if (type != null) {
    // Return "proper" type for logical operations! Previously this code assumed binary ops to be arithmetic,
    // and did not consider the proper type to return for the logical operators!
    switch (op) {
        case EQ:
        case NE:
        case LAND:
        case LOR:
        case LT:
        case GT:
        case LE:
        case GE:
            type = new BoolType();
            break;
    }
    debugStream.println("    Result of operation is of type " + type.getName());
}
return type;
}

```

```

/**
 * Validates the use of the binary boolean operation on the specified types. Line number and
 * column number are included for use in the error output (if needed). Returns a bool type
 * indicating the type of value returned from the expression.
 */
AbstractType checkBinaryBooleanOp(int op, AbstractType lType, AbstractType rType, int line, int col) {
    AbstractType type = new BoolType();

    if (rType != null && lType != null)
        debugStream.println("    Checking result of boolean operation between " + lType.getName() + " and " +
            rType.getName());
    else {
        debugStream.print("    Insufficient type information: (");
        if (lType != null)
            debugStream.print("LHS: " + lType);
        else
            debugStream.print("LHS: null");

        if (rType != null)
            debugStream.print("    RHS: " + rType);
        else
            debugStream.print("    RHS: null");
        debugStream.println(")");
    }
    try { // try and make the conversion
        typeManager.checkConversion(lType, type);
        typeManager.checkConversion(rType, type);
    } catch (IncompatibleTypesException ex) {
        reportError(lType.getName() + " " + rType.getName(), ErrorCodeTable.INCOMPATIBLE_TYPES, line, col);
        debugStream.println("    " + lType.getName() + " and " + rType.getName() +
            " are incompatible data types.");
    }
    return type;
}

/**
 * Validates the use of the unary boolean operation on the specified type. Line number and
 * column number are included for use in the error output (if needed). Returns a bool type
 * indicating the type of value returned from the expression.
 */
AbstractType checkUnaryBooleanOp(int op, AbstractType lType, int line, int col) {
    AbstractType type = new BoolType();

    if (lType != null)
        debugStream.println("    Checking result of boolean operation on " + lType.getName());
    else
        debugStream.println("    Insufficient type information: (" + lType + ")");

    if (lType.isStream()) // the type is a stream type
        return type;
    try { // try to perform a conversion to bool
        typeManager.checkConversion(lType, type);
    } catch (IncompatibleTypesException ex) {
        reportError(lType.getName(), ErrorCodeTable.INCOMPATIBLE_UNARY_BOOLEAN, line, col);
        debugStream.println("    A " + lType.getName() + " is an incompatible data type for a boolean operation.");
    }
    return type;
}

/**
 * Validates that the variable (value) for the function can be evaluated and is of the correct type.
 * The constantsAllowed parameter allows the function to check if the value is a constant, because
 * sometimes the variable cannot be a constant (I.E. - when the variable is being used as a return
 * value.
 */
public boolean validVariable(SandboxAST value, String type, boolean constantsAllowed) {
    try {
        // try to obtain the identifier's type
        String typeName = getType(value.getText());
        AbstractType variableType = typeManager.getType(typeName);

        if (variableType == null) {

```

```

    try {
        if (!constantsAllowed) // the variable can not be a constant
            return false;
        // check if the value is a constant
        variableType = constant(value);
        if (variableType == null) // the value is not an identifier or a constant
            return false;
        // else -- OK
    } catch (RecognitionException re) {
        //I dont kow what this does but it isnt good
        return false;
    }
}

if (!typeManager.checkConversion(variableType, typeManager.getType(type))) // incompatible type check
    return false;

return true;
} catch (IncompatibleTypesException ite) { // incompatible types
    return false;
}
}

/**
 * Builds the type manager's type map and locates/stores the shortest path from each type to
 * each type. Repeatedly prints out "declaring type" each time it is executed, which is generally
 * more than once (once for primitive types, once for included types).
 */
void assessTypes() {
    typeManager.buildTypeMap();
    for (Enumeration e = typeManager.getTypeNames(); e.hasMoreElements(); ) {
        java.lang.String typeName = (java.lang.String)e.nextElement();
        AbstractType type = typeManager.getType(typeName);

        if (!typesDeclared.contains(typeName))
        {
            typesDeclared.addElement(typeName);
            debugStream.println("Declaring type " + typeName);
        }

        TypeRepresentation rep = new TypeRepresentation(type);
        symtab.setRepresentationRecord(typeName, new RepresentationRecord());
        symtab.setRepresentation(typeName, rep);
    }
}

/**
 * Check to see if the test (used on if, for, while, and do statements) is semantically valid
 */
public boolean validTestExpression(AbstractType type) {
    if (type == null)
        return false;

    return (type.isNumeric() || type.isStream());
}

/**
 * Returns an array of class objects of actual parameter types to a library function call.
 * Actuals is assumed to be the ARGUMENTS node in the AST, so we first need to skip down
 * to the first parameter.
 */
protected Class[] getActualTypes(AST actuals) {
    if(actuals == null)
        return new Class[] { };

    AST param = (SandboxAST) actuals.getFirstChild();
    ArrayList tempList = new ArrayList();

    while (param != null) {
        try {
            AbstractType rep = expression(param);

```

```

// Handles the base types
if (rep == null) {
    reportError(param.getText() + " has not been declared." , ErrorCodeTable.NOT_DECLARED,
        ((SandboxAST) actuals).getLine(), 0);
    debugStream.println(" " + param.getText() + " not declared.");
}
else if (rep.isDoubleType() || rep.isDoubleLiteralType() || rep.isFloatType())
    tempList.add(DoubleRepresentation.class);

else if (rep.isIntType() || rep.isIntLiteralType() || rep.isShortType() || rep.isLongType())
    tempList.add(IntRepresentation.class);

else if (rep.isBoolType() || rep.isBoolLiteralType())
    tempList.add(BoolRepresentation.class);

else if (rep.isCharType() || rep.isCharLiteralType())
    tempList.add(CharRepresentation.class);

else if (rep.getName().equals("istream") || rep.getName().equals("ifstream"))
    tempList.add(IStreamRepresentation.class);

else if (rep.getName().equals("string"))
    tempList.add(sandbox.lib.string.StringRepresentation.class);

else if (rep.getName().equals("stringliteral"))
    tempList.add(StringLiteralRepresentation.class);

else
    debugStream.println(" unknown actual type: " + rep.getName());

    param = param.getNextSibling();
} catch (RecognitionException re) {
    re.printStackTrace();
}
}

// Build and return the list of class objects
int count = tempList.size();
Class[] classArray = new Class[count];
for (int i=0; i < count; i++)
    classArray[i] = (Class) tempList.get(i);
return classArray;
}

/**
 * Converts the src type (a Java class, as in Integer.class) to the corresponding
 * internal primitive type (IntType) for use in calculating type conversions when
 * looking up library function calls.
 */
public static AbstractType convertClassToAbstract(Class src) {
    if (src.getName().equals("java.lang.Integer") || src.getName().equals("int"))
        return (new IntType());
    else if (src.getName().equals("java.lang.Double"))
        return new DoubleType();
    else if (src.getName().equals("java.lang.Character"))
        return new CharType();
    else if (src.getName().equals("java.lang.Boolean"))
        return new BoolType();
    else if (src.getName().equals("sandbox.compiler.type.DoubleRepresentation"))
        return new DoubleType();
    else if (src.getName().equals("sandbox.compiler.type.IntRepresentation"))
        return new IntType();
    else
        return null; // this should never occur!
}
}

program
{
    debugStream.println("-----");
    debugStream.println("Checking program:");
    assessTypes();
}

```

```

}
: #(prog:PROGRAM (include)* (declaration | function | functionPrototype)*
{
  /**
   * AFTER checking all the declarations and functions,
   * we finally check if there is a main function.
   */
  if (fm.locateFunction("main", 0) == null) {
    reportError("No main function. Cannot proceed.",
      ErrorCodeTable.NO_MAIN, 0, 0);
    debugStream.println(" No main function. Cannot proceed.");
  }

  debugStream.println("-----");
  debugStream.println("Function hashtable contents:");
  debugStream.println(fm.toString());
}
;

include
: includeNode:INCLUDE
{
  AST name = includeNode.getFirstChild();

  // Since lexer already checks this, we can presume correctness
  try {

    if (name != null) {
      String incName = name.getText().trim();
      if (!restriction.isEnabled(Restriction.CLIMITS) && incName.equals("climits"))
        reportError ("climits", ErrorCodeTable.HEADER_DISABLED, ((SandboxAST) name).getLine(), 0);
      else if (!restriction.isEnabled(Restriction.CMATH) && incName.equals("cmath"))
        reportError ("cmath", ErrorCodeTable.HEADER_DISABLED, ((SandboxAST) name).getLine(), 0);
      else if (!restriction.isEnabled(Restriction.CSTDLIB) && incName.equals("cstdlib"))
        reportError ("cstdlib", ErrorCodeTable.HEADER_DISABLED, ((SandboxAST) name).getLine(), 0);
      else if (!restriction.isEnabled(Restriction.IOMANIP) && incName.equals("iomanip"))
        reportError ("iomanip", ErrorCodeTable.HEADER_DISABLED, ((SandboxAST) name).getLine(), 0);
      else if (!restriction.isEnabled(Restriction.IOSTREAM) && incName.equals("iostream"))
        reportError ("iostream", ErrorCodeTable.HEADER_DISABLED, ((SandboxAST) name).getLine(), 0);
      else if (!restriction.isEnabled(Restriction.FSTREAM) && incName.equals("fstream"))
        reportError ("fstream", ErrorCodeTable.HEADER_DISABLED, ((SandboxAST) name).getLine(), 0);
      else if (!restriction.isEnabled(Restriction.STRING) && incName.equals("string"))
        reportError ("string", ErrorCodeTable.HEADER_DISABLED, ((SandboxAST) name).getLine(), 0);
      else {
        Library lib = libraryManager.loadLibrary(incName);
        debugStream.println("-----");
        debugStream.println("Checking includes:");
        debugStream.println(" Loaded library " + incName);
        assessTypes();
        if (incName.equals("iostream"))
          lib.loadSymbols(symtab, null);
        else
          lib.loadSymbols(symtab);
      }
    }
  } catch (Exception ex) {
    reportError(ex.toString(), 0, 0, 0);
    debugStream.println(" Error loading library " + name.getText().trim());
    ex.printStackTrace(debugStream);
  }
}
;

functionPrototype
: pNode:FUNCTION_PROTOTYPE
{
  AST type = pNode.getFirstChild();
  AST name = type.getNextSibling();

  debugStream.println("-----");
  debugStream.println("Checking function prototype: " + name.getText());
  currentFunction = fm.addFunction(pNode, true);
  if (currentFunction.getParameterListRoot() != null) {
    ArrayList paramPassMechList = new ArrayList();

```

```

        AbstractType[] formals = formalParameters(currentFunction.getParameterListRoot(), false,
            paramPassMechList, ((SandboxAST) type).getLine());
        currentFunction.setFormalParams(formals, paramPassMechList);
    }
}
;

function
: funcNode:FUNCTION
{
    returnSeen = false;

    AST type = funcNode.getFirstChild();
    AST name = type.getNextSibling();
    AST tempNode = name.getNextSibling();

    debugStream.println("-----");
    debugStream.println("Checking function definition: " + name.getText());

    AST paramsNode = null;
    AST block = null;

    if (tempNode.getText().equals("PARAMETER_LIST")) {
        paramsNode = tempNode;
        block = paramsNode.getNextSibling();
    } else if (tempNode.getText().equals("BLOCK_STATEMENT"))
        block = tempNode;

    // Handle case of a function declaration, when we see a block_statement
    if (block != null) {
        currentFunction = fm.addFunction(funcNode, false);
        symtab.beginLevel();

        debugStream.println("  Checking formal parameters:");
        FunctionWrapper tempWrapper = new FunctionWrapper(funcNode, false);

        // Process the formal parameter list
        if (tempWrapper.getParameterListRoot() != null) {
            ArrayList paramPassMechList = new ArrayList();
            AbstractType[] formals = formalParameters(tempWrapper.getParameterListRoot(), true,
                paramPassMechList, ((SandboxAST) type).getLine());

            // Iterate through paramPassMechList and existing passMechList (from a prototype,
            // if any) and present errors, if any
            boolean foundError = false;
            if (paramPassMechList != null && !paramPassMechList.isEmpty()) {
                for (int index = 0; index < paramPassMechList.size(); index++) {
                    int functionMech = ((Integer) paramPassMechList.get(index)).intValue();
                    int protoMech = currentFunction.getFormParamPassMech(index);
                    if (protoMech != FunctionWrapper.INVALID && protoMech != functionMech) {
                        reportError(tempWrapper.getParamName(index), ErrorCodeTable.PASSING_MISMATCH,
                            ((SandboxAST) type).getLine(), 0);
                        debugStream.println("Function parameter '" + tempWrapper.getParamName(index) +
                            "' does not match its prototype passing mechanism.");
                        foundError = true;
                    }
                }
            }
            if (!foundError)
                currentFunction.setFormalParams(formals, paramPassMechList);
        }

        // Check the return type (does it match the prototype, if a proto exists?)
        if (currentFunction.getPrototypeSeen()) {
            AbstractRepresentation rep = symtab.getRepresentation(type.getText());

            if (rep != null && rep.isTypeRepresentation())
                if (!type.getText().equals(currentFunction.getReturnType())) {
                    reportError(type.getText() + " " + currentFunction.getReturnType(),
                        ErrorCodeTable.FUNCTION_TYPE_MISMATCH, ((SandboxAST) type).getLine(), 0);
                    debugStream.println("Function return type (" + type.getText() + ") does not match prototype return type (" +
                        currentFunction.getReturnType() + ").");
                }
        }
    }
}

```

```

        debugStream.println("    Checking function body");
        blockWithoutNewAR(block);
        symtab.endLevel();

        // Process the return type (if present)
        debugStream.println("    Validating presence of return statement, if needed.");
        if (!returnSeen && !type.getText().equals("void")) {
            reportError("Function " + name.getText() + " returns a " + type.getText() +
                " type.", ErrorCodeTable.MISSING_RETURN, ((SandboxAST) type).getLine(), 0);
            debugStream.println("Function '" + name.getText() + "' fails to return a '" +
                type.getText() + "' type.");
        }

        debugStream.println("End of function check.");
    }
}
;

formalParameters [boolean isFunction, ArrayList paramPassMech, int lineNum] returns [AbstractType[] formalParams]
{
    // The 'isFunction' parameter indicates whether (true) or not (false) we are processing
    // a function definition or a function prototype.

    formalParams = null;
}
: parameterListNode:PARAMETER_LIST
{
    Iterator iter;
    int index;
    ArrayList formalParamsList = new ArrayList();

    for (AST node = parameterListNode.getFirstChild(); node != null; node = node.getNextSibling()) {
        formalParamsList.add(formalParameter(node, isFunction, lineNum));

        AST passMechanism = null;
        boolean isConst = false;

        if (node.getFirstChild().getText().equals("TYPE_MODIFIER")) {
            if (node.getFirstChild().getFirstChild().getText().equals("const"))
                isConst = true;
            passMechanism = node.getFirstChild().getNextSibling().getNextSibling();
        } else
            passMechanism = node.getFirstChild().getNextSibling();

        if (passMechanism != null && passMechanism.getText().equals("&"))
            if (!isConst)
                paramPassMech.add(new Integer(FunctionWrapper.BY_REF));
            else
                paramPassMech.add(new Integer(FunctionWrapper.BY_CONST_REF));
        else
            if (!isConst)
                paramPassMech.add(new Integer(FunctionWrapper.BY_VALUE));
            else
                paramPassMech.add(new Integer(FunctionWrapper.BY_CONST_VALUE));
    }
    formalParams = new AbstractType[formalParamsList.size()];
    for (index = 0, iter = formalParamsList.iterator(); iter.hasNext();
        formalParams[index++] = (AbstractType) iter.next();
    )
}
;

formalParameter [boolean isFunction, int lineNum] returns [AbstractType type]
{
    // The 'isFunction' parameter indicates whether or not we are processing
    // a function definition (true) or a function prototype (false).

    type = null;
}
: formalNode:PARAMETER
{
    AST typeNode = formalNode.getFirstChild();
    boolean isConst = false;
    boolean passedByReference = false;
}
;

```

```

// Handle the processing of type modifiers (const)
if (typeNode.getText().equals("TYPE_MODIFIER")) {
    if (typeNode.getFirstChild().getText().equals("const"))
        isConst = true;
    typeNode = typeNode.getNextSibling();
}
debugStream.println("    parameter located: " + typeNode.getText());

// Attempt to obtain the parameter's name
AST paramNode = typeNode.getNextSibling();

// If this is a reference parameter, skip the token and pick up the parameter's name
if (paramNode != null && paramNode.getText().equals("&")) {
    passedByReference = true;
    paramNode = paramNode.getNextSibling();
}

// Check if the parameter exists already in the symbol table.
if (paramNode != null && symtab.isLocal(paramNode.getText())) {
    reportError(paramNode + " already declared!", ErrorCodeTable.REDECLARED, lineNum, 0);
    debugStream.println(paramNode + " already declared!");
} else {
    AbstractRepresentation rep = symtab.getRepresentation(typeNode.getText());

    if (rep != null && rep.isTypeRepresentation()) {
        type = ((TypeRepresentation) rep).getRepresentedType();
        RepresentationRecord newRec = new RepresentationRecord();
        if (isConst)
            newRec.setConstant(true);

        // Ensure streams are passed by reference!
        if (type.isStream() && !passedByReference) {
            reportError(ErrorCodeTable.STREAMS_NOT_PASSED_BY_REF, lineNum, 0);
            debugStream.println("    A file stream parameter was not passed by reference.");
        }

        // Only add function definition formal parameters to the symbol table!
        if (paramNode != null) {
            if (isFunction) {
                symtab.setRepresentationRecord(paramNode.getText(), newRec);
                symtab.setRepresentation(paramNode.getText(), type.instantiate());
            }
            else if (isFunction) {
                reportError(ErrorCodeTable.MISSING_IDENTIFIER_IN_PARAM_LIST, lineNum, 0);
                debugStream.println("    An identifier is missing from the parameter list.");
            }
            else {
                SandboxAST fNode = (SandboxAST) formalNode;
                reportError(typeNode.getText() + " is not a valid datatype.", ErrorCodeTable.NOT_TYPE, fNode.getLine(), 0);
                debugStream.println(typeNode.getText() + " is not a valid datatype.");
            }
        }
    }
}
}
;

/**
 * STATEMENTS PART
 */
statement
: whileStatement
| forStatement
| doStatement
| ifStatement
| switchStatement
| expressionStatement
| blockStatement
| declaration
| returnStatement
| breakStatement
| continueStatement
;

switchStatement

```

```

{
debugStream.println("Checking switch statement");
}
: #(switchNode:SWITCH (exprNode:IDENTIFIER
{
// add a new ArrayList to the switchLabelStack (for the switch statement currently being checked)
switchLabelStack.push(new ArrayList());

// Validate that the switch expression is a char, int, charLiteral, or intLiteral
java.lang.String identifType = getType(exprNode.getText());

if (identifType == null)
reportError(exprNode.getText(), ErrorCodeTable.NOT_DECLARED, ((SandboxAST) exprNode).getLine(), 0);

else if (!identifType.equals("int") && !identifType.equals("char"))
reportError(exprNode.getText() + " is not integer/character; cannot be a switch variable.",
ErrorCodeTable.INVALID_TYPE_SWITCH_IDENTIFIER, ((SandboxAST) switchNode).getLine(), 0);

AST caseGroupNode = exprNode.getNextSibling();
while (caseGroupNode != null) {
casesGroup(caseGroupNode);
caseGroupNode = caseGroupNode.getNextSibling();
}

// Remove the current ArrayList (of case labels), since we're done processing this switch statement
switchLabelStack.pop();
}
| (PLUS | MINUS | STAR | SLASH | PERCENT)
{
// This is a basic expression, let's let it pass and let the interp handle it. These may produce non-int(long) results,
// while may give the interpreter fits, so something here needs to be added to ensure the int result.
AbstractType exprType = expression(switchNode.getFirstChild());
if (!(exprType.isIntType()))
reportError("", ErrorCodeTable.SWITCH_EXPRESSION_NOT_INT_RESULT, ((SandboxAST) switchNode).getLine(), 0);
}
| iNode:INTEGER_CONSTANT | cNode:CHAR_CONSTANT | fNode:FLOATING_CONSTANT | sNode:STRING_LITERAL
| tNode:TRUE | flNode:FALSE)
{
// add a new ArrayList to the switchLabelStack (for the switch statement currently being checked)
switchLabelStack.push(new ArrayList());
AST caseGroupNode = null;

if (iNode != null) {
reportError(iNode.getText() + " is not integer/character; cannot be a switch variable.",
ErrorCodeTable.INVALID_TYPE_SWITCH_IDENTIFIER, ((SandboxAST) switchNode).getLine(), 0);
caseGroupNode = iNode.getNextSibling();
} else if (cNode != null) {
reportError(cNode.getText() + " is not integer/character; cannot be a switch variable.",
ErrorCodeTable.INVALID_TYPE_SWITCH_IDENTIFIER, ((SandboxAST) switchNode).getLine(), 0);
caseGroupNode = cNode.getNextSibling();
} else if (fNode != null) {
reportError(fNode.getText() + " is not integer/character; cannot be a switch variable.",
ErrorCodeTable.INVALID_TYPE_SWITCH_IDENTIFIER, ((SandboxAST) switchNode).getLine(), 0);
caseGroupNode = fNode.getNextSibling();
} else if (sNode != null) {
reportError(sNode.getText() + " is not integer/character; cannot be a switch variable.",
ErrorCodeTable.INVALID_TYPE_SWITCH_IDENTIFIER, ((SandboxAST) switchNode).getLine(), 0);
caseGroupNode = sNode.getNextSibling();
} else if (tNode != null) {
reportError(tNode.getText() + " is not integer/character; cannot be a switch variable.",
ErrorCodeTable.INVALID_TYPE_SWITCH_IDENTIFIER, ((SandboxAST) switchNode).getLine(), 0);
caseGroupNode = tNode.getNextSibling();
} else if (flNode != null) {
reportError(flNode.getText() + " is not integer/character; cannot be a switch variable.",
ErrorCodeTable.INVALID_TYPE_SWITCH_IDENTIFIER, ((SandboxAST) switchNode).getLine(), 0);
caseGroupNode = flNode.getNextSibling();
}

while (caseGroupNode != null) {
casesGroup(caseGroupNode);
caseGroupNode = caseGroupNode.getNextSibling();
}

// Remove the current ArrayList (of case labels), since we're done processing this switch statement

```

```

        switchLabelStack.pop();
    }
    ;

casesGroup
{
    AST sList = null;
}
: #(CASE_GROUP (acase)+ sList=caseSList)
;

acase
{
    AbstractType type;
    ArrayList currentLabelList = (ArrayList) switchLabelStack.peek();
}
: #(caseNode:CASE type=expression)
{
    // Check to see if the label is charLiteral or intLiteral
    AST caseLabel = caseNode.getFirstChild();
    if (currentLabelList.contains(caseLabel.getText()))
        reportError(caseLabel.getText(), ErrorCodeTable.DUPLICATED_CASE, ((SandboxAST) caseNode).getLine(), 0);
    else {
        if (getType(caseLabel.getText()) == null) {
            // The type of the constant (literal) is returned here
            AbstractType labelType = constant(caseLabel);

            // If the constant is not an integer or character an error is thrown
            if ((labelType != null) && ((labelType.isIntLiteralType() || (labelType.isCharLiteralType()))))
                currentLabelList.add(caseLabel.getText());
            else
                reportError(caseLabel.getText(), ErrorCodeTable.INVALID_CASE, ((SandboxAST) caseLabel).getLine(), 0);
        } else {
            // Case label had a type so it's likely an identifier... but what kind?
            String ident = identifier(caseLabel);
            if ((ident != null) && (symtab.getRepresentationRecord(ident).isConstant())) {
                if (!getType(ident).equals("int") && !getType(ident).equals("char"))
                    reportError(ident, ErrorCodeTable.INVALID_CASE, ((SandboxAST) caseNode).getLine(), 0);
                else
                    currentLabelList.add(caseLabel.getText());
            } else
                reportError(caseNode.getText(), ErrorCodeTable.INVALID_CASE, ((SandboxAST) caseNode).getLine(), 0);
        }
    }
}
| defNode:DEFAULT
{
    // We can have only one default in any switch statement. The first one seen is added
    // to the currentLabelList. If encountered again, then an error is thrown.
    if (!currentLabelList.contains("default"))
        currentLabelList.add("default");
    else
        reportError("", ErrorCodeTable.TOO_MANY_SWITCH_DEFAULTS, ((SandboxAST) defNode).getLine(), 0);
}
;

caseSList returns [AST firstStatement]
{
    firstStatement = null;
}
: #(sNode:SLIST (statement)+)
{
    // Nothing to check here, just let them trickle into the statement checker
    if (firstStatement == null)
        firstStatement = sNode.getFirstChild();
}
;

returnStatement
{

```

```

    AbstractType type=null;
}
: #(returnNode:RETURN (type=expression)?)
{
    debugStream.println("    Checking the return type (" + currentFunction.getName() + ")");
    debugStream.println("        Return type is: " + currentFunction.getReturnType());

    try {
        returnSeen = true;
        AbstractType typeToReturn = ((TypeRepresentation) symtab.getRepresentation(
            currentFunction.getReturnType()).getRepresentedType());
        if (!typeToReturn.isVoidType() && type == null) {
            reportError("null " + currentFunction.getReturnType(), ErrorCodeTable.RETURN_TYPE_MISMATCH,
                ((SandboxAST) returnNode).getLine(), 0);
            debugStream.println("        " + currentFunction.getReturnType() + " and *null* are incompatible data types");
        } else
            typeManager.checkConversion(typeToReturn, type);
    }
    catch (IncompatibleTypesException ite) {
        reportError(type.getName() + " " + currentFunction.getReturnType(), ErrorCodeTable.RETURN_TYPE_MISMATCH,
            ((SandboxAST) returnNode).getLine(), 0);
        debugStream.println("        " + currentFunction.getReturnType() + " and " + type.getName() +
            " are incompatible data types");
    }
};

breakStatement
: BREAK
;

continueStatement
: CONTINUE
;

blockStatement
{
    symtab.beginLevel();
}
: #(BLOCK_STATEMENT (statement)*)
{
    symtab.endLevel();
};

// This was put in to handle the case of function defitions where we don't want to create a new AR
// twice when we see a function def, but we want to create a new one up in functionDef so that the
// formal params make it into the correct AR.
blockWithoutNewAR
: #(BLOCK_STATEMENT (statement)*)
;

forStatement
{
    AbstractType initNode=null;
    AbstractType condNode=null;
    AbstractType iterNode=null;
}
: #(forNode:FOR ((initNode=forInit | declaration) condNode=forCond iterNode=forIter (statement)??) {
    if (condNode != null && !validTestExpression(condNode))
        reportError("", ErrorCodeTable.INVALID_BOOLEAN_EXPRESSION, ((SandboxAST) forNode).getLine(), 0);
}
);

forInit returns [AbstractType exprNode]
{
    exprNode=null;
}
: #(forinit:FOR_INIT (exprNode=expression)?)
;

```

```

forCond returns [AbstractType exprNode]
{
  exprNode=null;
}
: #(forcond:FOR_COND (exprNode=expression)?)
;

forIter returns [AbstractType exprNode]
{
  exprNode=null;
}
: #(forincr:FOR_ITER (exprNode=expression)?)
;

ifStatement
{
  AbstractType type=null;
}
: #(ifNode:IF type=expression statement (statement)?) {
  if (!validTestExpression(type)) {
    reportError("", ErrorCodeTable.INVALID_BOOLEAN_EXPRESSION, ((SandboxAST)ifNode).getLine(), 0);
  }
}
;

whileStatement
{
  AbstractType type=null;
}
: #(whileNode:WHILE type=expression (statement)?) {
  if (!validTestExpression(type)) {
    reportError("", ErrorCodeTable.INVALID_BOOLEAN_EXPRESSION, ((SandboxAST)whileNode).getLine(), 0);
  }
}
;

doStatement
{
  AbstractType type=null;
}
: #(doNode:DO blockStatement type=expression) {
  if (!validTestExpression(type)) {
    reportError("", ErrorCodeTable.INVALID_BOOLEAN_EXPRESSION, ((SandboxAST)doNode).getLine(), 0);
  }
}
;

expressionStatement
{
  AbstractType type=null;
}
: #(EXPRESSION_STATEMENT type=expression)
;

/*-----
DECLARATION PART
-----*/

declaration
: declNode:DECLARATION
{
  AST modifierNode = declNode.getFirstChild();
  AST typeNode;

  if (modifierNode.getText().equals("USING_DIRECTIVE")) {
    // Reads the name of the namespace as the "typeNode"
    // Just a hack so that we can recognize the 'using' line and

```

```

    // mark that that the "correct" namespace (std) is in use
    typeNode = modifierNode.getFirstChild();
    if (typeNode.getText().equals("std"))
        usingStdSet = true;
} else {
    if (modifierNode.getText().equals("TYPE_MODIFIER"))
        typeNode = modifierNode.getNextSibling();
    else {
        typeNode = modifierNode;
        modifierNode = null;
    }

    AST declaratorNode = typeNode.getNextSibling();
    SandboxAST mNode = (SandboxAST) typeNode;
    if (declaratorNode == null)
        reportError("Declarator is missing :", ErrorCodeTable.MISSING_DECLARATOR, mNode.getLine(), 0);
    else {
        String dataType = typeNode.getText();

        // Check restrictions on the primitive types!
        if (!restriction.isEnabled(Restriction.BOOLEAN) && dataType.equals("bool"))
            reportError ("boolean", ErrorCodeTable.IDENTIFIER_DISABLED, mNode.getLine(), 0);
        else if (!restriction.isEnabled(Restriction.CHAR) && dataType.equals("char"))
            reportError ("character", ErrorCodeTable.IDENTIFIER_DISABLED, mNode.getLine(), 0);
        else if (!restriction.isEnabled(Restriction.SHORT) && dataType.equals("short"))
            reportError ("short", ErrorCodeTable.IDENTIFIER_DISABLED, mNode.getLine(), 0);
        else if (!restriction.isEnabled(Restriction.INT) && dataType.equals("int"))
            reportError ("integer", ErrorCodeTable.IDENTIFIER_DISABLED, mNode.getLine(), 0);
        else if (!restriction.isEnabled(Restriction.LONG) && dataType.equals("long"))
            reportError ("long", ErrorCodeTable.IDENTIFIER_DISABLED, mNode.getLine(), 0);
        else if (!restriction.isEnabled(Restriction.FLOAT) && dataType.equals("float"))
            reportError ("float", ErrorCodeTable.IDENTIFIER_DISABLED, mNode.getLine(), 0);
        else if (!restriction.isEnabled(Restriction.DOUBLE) && dataType.equals("double"))
            reportError ("double", ErrorCodeTable.IDENTIFIER_DISABLED, mNode.getLine(), 0);

        // Ensure namespace 'std' is in use for strings, ifstream, and ofstream, if applicable.
        if (dataType.equals("string") || dataType.equals("ifstream") || dataType.equals("ofstream"))
            // Enforce 'using namespace std' when using cout or file streams!
            if (!usingStdSet && !usingErrMsgShown) {
                reportError("Namespace 'std' not in use.", ErrorCodeTable.STD_NOT_IN_USE,
                    ((SandboxAST) typeNode).getLine(), 0);
                usingErrMsgShown = true;
            }

        AbstractRepresentation rep = syntab.getRepresentation(dataType);
        if ((rep != null) && (rep.isTypeRepresentation()) && (!typeNode.getText().equals("void"))) {
            while (declaratorNode != null) {
                declarator(declaratorNode, typeNode.getText(), modifierNode);
                declaratorNode = declaratorNode.getNextSibling();
            }
        } else {
            reportError(dataType + " is not a valid data type.", ErrorCodeTable.NOT_TYPE, mNode.getLine(), 0);
            debugStream.println(dataType + " is not a valid data type.");
        }
    }
}
}
}
;

declarator [java.lang.String typeName, AST modNode]
{
    SandboxAST paramsNode= null;
    AbstractType type = null;
}
// VARIABLE declaration
: typeN:TYPENAME
{
    reportError(typeN.getText() + " is an invalid identifier name.", ErrorCodeTable.INVALID_IDENTIFIER_NAME,
        ((SandboxAST) typeN).getLine(), ((SandboxAST) typeN).getLine());
}
| #(declNode:DECLARATOR (id:IDENTIFIER ((type=expression)? | paramsNode=declareParameters)
{
    if (id != null) {

```



```

        }
        else {
            reportError("", ErrorCodeTable.INVALID_ARRAY_SIZE, ((SandboxAST) identifNode).getLine(), 0);
        }
    }
    else {
        reportError("", ErrorCodeTable.ARRAY_INITIALIZE, ((SandboxAST) identifNode).getLine(), 0);
    }
}
}
;

typeModifier [java.lang.String varname]
: typeMod:TYPE_MODIFIER
{
    SandboxAST modifierNode = (SandboxAST) typeMod.getFirstChild();
    if (modifierNode.getText().equals("const")) {
        RepresentationRecord r = symtab.getRepresentationRecord(varname);
        r.setConstant(true);
    }
    else {
        reportError("Don't know what to do with " + modifierNode.getText(), ErrorCodeTable.SEMANTIC_ERROR,
            modifierNode.getLine(), 0 );
    }
}
;

declareParameters returns [SandboxAST pList]
{
    pList = null;
}
: #(paramList:PARAMETER_LIST (declareParameter)*)
{
    pList = (SandboxAST) paramList;
}
;

declareParameter
: formalNode:PARAMETER
{
    AST typeNode = formalNode.getFirstChild();
    if (typeNode.getText().equals("TYPE_MODIFIER")) {
        typeNode = typeNode.getNextSibling();
    }

    AbstractRepresentation rep = symtab.getRepresentation(typeNode.getText());

    if (rep == null || !(rep.isTypeRepresentation())) {
        SandboxAST tNode = (SandboxAST) typeNode;
        reportError(typeNode.getText() + " is not a valid datatype.", ErrorCodeTable.NOT_TYPE, tNode.getLine(), 0);
        debugStream.println(typeNode.getText() + " is not a valid datatype.");
    }
}
;

/*-----
EXPRESSION PART
-----*/

primaryExpr returns [AbstractType type]
{
    SandboxAST name;
    java.lang.String IDName;

    if (_t != null)
        debugStream.println("    Evaluating primaryExpr:" + _t.toStringList());
    else
        debugStream.println("    Evaluating primaryExpr: *** expression is null***");
    type = null;
}
: IDName=identifier

```

```

    {
        if (IDName != null) {
            AbstractRepresentation rep = syntab.getRepresentation(IDName);
            type = rep.getType();
        }
    }

    | type=constant
    | #(LPAREN type=expression)
    ;

identifier returns [java.lang.String type]
{
    type = null;
}
: idval:IDENTIFIER
{
    AbstractRepresentation rep = syntab.getRepresentation(idval.getText());

    debugStream.println("    Checking identifier: " + idval.getText());
    if (rep == null) {
        SandboxAST iNode = (SandboxAST) idval;

        reportError(idval.getText() + " has not been declared." ,
            ErrorCodeTable.NOT_DECLARED, iNode.getLine(), 0);
        debugStream.println("        " + idval.getText() + " not declared.");
    }
    // Check for identifier.field case. This check was inserted to keep Sandbox
    // from allowing code like int x; x.foo; to pass through. If we ever decide to
    // implement structs, this check will need to be removed and a better workaround
    // found.
    else if (idval.getNextSibling() != null && idval.getNextSibling().getText().equals(".")) {
        SandboxAST iNode = (SandboxAST) idval;
        reportError(idval.getText() + "." + idval.getNextSibling().getNextSibling().getText() +
            " has not been declared.", ErrorCodeTable.NOT_DECLARED, iNode.getLine(), 0);
        debugStream.println("        " + idval.getText() + "." +
            idval.getNextSibling().getNextSibling().getText() + " not declared.");
    }
    else {
        debugStream.println("    Identifier is of type " + rep.getType().getName());
        type = idval.getText();
    }
}
;

constant returns [AbstractType type]
{
    type = null;
}
: INTEGER_CONSTANT
  { type = new IntLiteralType(); }
| FLOATING_CONSTANT
  { type = new DoubleLiteralType(); }
| CHAR_CONSTANT
  { type = new CharLiteralType(); }
| STRING_LITERAL
  { type = new StringLiteralType(); }
| ( TRUE | FALSE )
  { type = new BoolLiteralType(); }
;

expression returns [AbstractType type]
{
    SandboxAST name, name2 = null, name3=null, name4=null;
    AbstractType lType=null, rType=null, negType, lType2, lType3, cons;
    java.lang.String leftID;

    if (_t != null)
        debugStream.println("Checking expression: " + _t.toStringList());
    else
        debugStream.println("Checking expression: ***expression is null***");
    type = null;
}
: callNode:CALL

```

```

{
    name = (SandboxAST) callNode.getFirstChild();
    java.lang.String streamName = name.getText();
    java.lang.String typeName = getType(streamName);
    java.lang.String functionName = streamName;

    if (typeName != null && (typeName.equals("ofstream") || typeName.equals("ifstream"))) {
        // Enforce 'using namespace std' when using file streams!
        if (!usingStdSet && !usingErrMsgShown) {
            reportError("Namespace 'std' not in use.", ErrorCodeTable.STD_NOT_IN_USE,
                ((SandboxAST) name).getLine(), 0);
            usingErrMsgShown = true;
        }

        // Reads and ignores the 'dot' in the tree
        name = (SandboxAST) name.getNextSibling();
        // Reads the name of the method call
        name = (SandboxAST) name.getNextSibling();
        //
        // OPEN() method call
        //
        if (name.getText().equals("open")) {
            name2 = (SandboxAST) name.getNextSibling();
            if (name2 == null)
                reportError(name.getText(), ErrorCodeTable.ARGUMENT_NULL, name.getLine(), 0);
            else {
                name2 = (SandboxAST) name2.getFirstChild();
                name3 = (SandboxAST) name2.getNextSibling();
                if (name3 != null)
                    reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);

                String argument = name2.getText();
                int invalidCh = '/';
                int invalidChar = '\\';

                if (argument.equals("") || argument.indexOf(invalidCh) != -1 || argument.indexOf(invalidChar) != -1)
                    reportError(name.getText(), ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);
                else {
                    // Obtain a handle to the representation of the stream and call setOpened method
                    if (typeName.equals("ofstream")) {
                        OFStreamRepresentation fstreamRep = (OFStreamRepresentation) symtab.getRepresentation(streamName);
                        fstreamRep.setOpened(true);
                    } else {
                        IFStreamRepresentation fstreamRep = (IFStreamRepresentation) symtab.getRepresentation(streamName);
                        fstreamRep.setOpened(true);
                    }
                }
            }
        }
        //
        // CLOSE() method call
        //
        } else if (name.getText().equals("close")) {
            name2 = (SandboxAST) name.getNextSibling();
            if (name2 != null)
                reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
            else {
                // Obtain a handle to the representation of the stream and call setOpened method
                if (typeName.equals("ofstream")) {
                    OFStreamRepresentation fstreamRep = (OFStreamRepresentation) symtab.getRepresentation(streamName);
                    fstreamRep.setOpened(false);
                } else {
                    IFStreamRepresentation fstreamRep = (IFStreamRepresentation) symtab.getRepresentation(streamName);
                    fstreamRep.setOpened(false);
                }
            }
        }
        //
        // FAIL() method call
        //
        else if (name.getText().equals("fail")) {
            name2 = (SandboxAST) name.getNextSibling();
            if (name2 != null)
                reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
        }
    }
}

```

```

    type = new BoolType();
}
//
// CLEAR() method call
//
else if (name.getText().equals("clear") )
{
    name2=(SandboxAST) name.getNextSibling();
    if (name2 != null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
}
//
// EOF() method call
//
else if (name.getText().equals("eof") )
{
    name2=(SandboxAST) name.getNextSibling();
    if (name2 != null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);

    type= new BoolType();
}
//
// PEEK() method call
//
else if (name.getText().equals("peek") ) {
    name2=(SandboxAST) name.getNextSibling();
    if (name2 != null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);

    type= new IntType();
}
//
// IGNORE() method call
//
else if (name.getText().equals("ignore") && typeName.equals("ifstream")) {
    name2 = (SandboxAST) name.getNextSibling();
    if (name2 == null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
    else {
        // Check for parameters of ignore now.
        name2 = (SandboxAST) name2.getFirstChild();
        name3 = (SandboxAST) name2.getNextSibling();

        // Check the first parameter
        AbstractType exprType = expression(name2);
        if (exprType == null || !(exprType.isIntType()) && !(exprType.isIntLiteralType()))
            reportError(functionName, ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);

        // Check if the second paramter exists, if so then check if it is valid
        if (name3 != null) {
            // Ignore call has 2 parameters

            // Check for a 3rd parameter
            name4 = (SandboxAST) name3.getNextSibling();
            if (name4 != null)
                reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
            else
                // Check 2nd parameter validity
                exprType = expression(name3);
                if (!(exprType.isCharType()) && !(exprType.isCharLiteralType()))
                    reportError(functionName, ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);
        }
    }
}
//
// GET() method call
//
else if (name.getText().equals("get")) {
    name2=(SandboxAST) name.getNextSibling();

    if (name2 == null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
    else {

```

```

    // Check for parameter type of get now...should be a character
    name2 = (SandboxAST) name2.getFirstChild();
    name3 = (SandboxAST) name2.getNextSibling();
    if (name3 != null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
    if (!validVariable(name2, "char", false))
        reportError(name.getText(), ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);
}
}
//
// PUTBACK() method call
//
else if (name.getText().equals("putback")) {
    name2=(SandboxAST) name.getNextSibling();

    if (name2 == null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
    else {
        // Check for parameter type of get/putback now...should be a character
        name2 = (SandboxAST) name2.getFirstChild();
        name3 = (SandboxAST) name2.getNextSibling();
        java.lang.String identifType = getType(name2.getText());

        if (name2 == null || name3 != null)
            reportError(name.getText(), ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
        if (!validVariable(name2, "char", true) && !validVariable(name2, "charliteral", true))
            reportError(name.getText(), ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);
    }
}
else {
    reportError(name.getText(), ErrorCodeTable.UNABLE_TO_LOCATE_FUNCTION,
        ((SandboxAST) name).getLine(), 0);
    debugStream.println(" " + name.getText() + " function not found, possible scope issue...");
}
} else if (typeName != null && typeName.equals("istream")) {
    // Reads and ignores the 'dot' in the tree
    name = (SandboxAST) name.getNextSibling();

    // Reads the name of the method call
    name = (SandboxAST) name.getNextSibling();

    // Enforce 'using namespace std' when using input streams!
    if (!usingStdSet && !usingErrMsgShown) {
        reportError("Namespace 'std' not in use.", ErrorCodeTable.STD_NOT_IN_USE,
            ((SandboxAST) name).getLine(), 0);
        usingErrMsgShown = true;
    }
}

if (name.getText().equals("get")) {
    // Obtain the number and type of arguments...
    name2=(SandboxAST) name.getNextSibling();

    if (name2 == null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
    else {
        // check for parameter type of get now...should be a character
        name2 = (SandboxAST) name2.getFirstChild();
        name3 = (SandboxAST) name2.getNextSibling();
        if (name3 != null)
            reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
        if (!validVariable(name2, "char", false))
            reportError(name.getText(), ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);
    }
}
} else if (name.getText().equals("ignore")) {
    name2=(SandboxAST) name.getNextSibling();
    if (name2 == null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
    else {
        // Check for parameters of ignore now.
        name2 = (SandboxAST) name2.getFirstChild();
        name3 = (SandboxAST) name2.getNextSibling();

        // Check the first parameter
        AbstractType exprType = expression(name2);

```



```

        debugStream.println(" " + actualParams[index].getName() +
            " cannot be coerced to " + formalParams[index].getName());
    }
}
type = ((TypeRepresentation) sytab.getRepresentation(function.getReturnType()).getRepresentedType());
} else if (formalParamCount > 0) {
    reportError("Number of parameters do not match. Function '" + funcName + "' expects " + formalParamCount +
        " arguments.", ErrorCodeTable.PARAM_COUNT_MISMATCH, ((SandboxAST) name).getLine(), 0);
    debugStream.println("Parameter count mismatch in function '" + funcName + "'.");
}
} else if (libraryManager.containsFunction(funcName)) {
    // The call is to a library function
    debugStream.println("Calling library function: " + funcName);

    // Ensure what's in the library is call-able, otherwise, it's likely to be
    // handled here in the interpreter, above, as a special function (for I/O)!
    if (libraryManager.inLibrary(funcName)) {
        Class libClass = null;
        Class[] actTypes = getActualTypes(name.getNextSibling());

        try {
            libClass = Class.forName(libraryManager.getOwner(funcName));
            Method libMethod = libClass.getMethod(funcName, actTypes);
            // This is a major hack and a bad way to test the constant modification, but how
            // do we do this inside
            // the library. That is, we can check the situation, but how do we create and
            // report the error from in there?
            // Likely, we'd need to throw an exception and then try to catch it out here...
            // think in the big picture for
            // other situations like this, can we do that for those as well?
            if (funcName.equals("getline")) {
                String secondParam = name.getNextSibling().getFirstChild().getNextSibling().getText();
                if (sytab.getRepresentationRecord(secondParam).isConstant()) {
                    reportError(secondParam, ErrorCodeTable.ASSIGNMENT_TO_CONSTANT,
                        ((SandboxAST) name).getLine(), 0);
                    debugStream.println(" " + secondParam + " is a constant, and can not be modified.");
                }
            }
        }
    }
} catch (NoSuchMethodException nsme) {
    // OK, we didn't find an exact match for the intended function call,
    // let's check for a close match (look for one with the same name and same number
    // of params (and try to coerce)
    debugStream.println(" Direct function match not found, searching for close match...");
    Method[] libMethods = libClass.getMethods();
    debugStream.println(" Checking through " + libMethods.length + " methods of " +
        libClass.getName());
    debugStream.println(" Looking for a match with " + actTypes.length + " parameter(s).");
    int bestFitIndex = -1;
    int leastConversions = 999;
    for (int index=0; index < libMethods.length; index++) {
        // Since we get ALL public methods, we need to check both the names and the paramlist size
        if (libMethods[index].getName().equals(funcName) &&
            libMethods[index].getParameterTypes().length == actTypes.length) {
            // Start worrying finding each possible matching function here
            Class[] libParamTypes = libMethods[index].getParameterTypes();
            debugStream.println(" Function match found (based on name): " +
                libParamTypes.length + " param(s).");
            int sumConversions = 0;
            for (int paramIndex = 0; paramIndex < libParamTypes.length; paramIndex++)
                sumConversions += typeManager.checkConversionSteps(
                    NewSemantic.convertClassToAbstract(libParamTypes[paramIndex]),
                    NewSemantic.convertClassToAbstract(actTypes[paramIndex]));

            debugStream.println(" The number of conversions for all parameters is: " + sumConversions);
            // Keep a reference to the best one match found
            if (sumConversions > 0 && sumConversions < leastConversions) {
                leastConversions = sumConversions;
                bestFitIndex = index;
            }
        }
    }
}
}
}

```



```

    if (!(syntab.getRepresentationRecord(leftID).isConstant())) {
        AbstractRepresentation rep = syntab.getRepresentation(leftID);

        if (checkBinaryOp (MINUS_ASSIGN, rep.getType(), rType, ((SandboxAST) mAssignNode).getLine(), 0) != null)
            type = checkAssignment (MINUS_ASSIGN, rep.getType(), rType, ((SandboxAST) mAssignNode).getLine(), 0);
        } else
            reportError(leftID + " is a constant; it cannot be modified.",
                ErrorCodeTable.ASSIGNMENT_TO_CONSTANT,((SandboxAST) mAssignNode).getLine(),0);
    })
}
| #({sAssignNode:STAR_ASSIGN (lType=constant
{
    if (lType != null)
        reportError("", ErrorCodeTable.LITERAL_ASSIGN, ((SandboxAST) sAssignNode).getLine(), 0);
}
| leftID=identifier rType=expression
{
    if (leftID != null)
        if (!(syntab.getRepresentationRecord(leftID).isConstant())) {
            AbstractRepresentation rep = syntab.getRepresentation(leftID);

            if (checkBinaryOp (STAR_ASSIGN, rep.getType(), rType, ((SandboxAST) sAssignNode).getLine(), 0) != null)
                type = checkAssignment (STAR_ASSIGN, rep.getType(), rType, ((SandboxAST) sAssignNode).getLine(), 0);
            } else
                reportError(leftID + " is a constant; it cannot be modified.",
                    ErrorCodeTable.ASSIGNMENT_TO_CONSTANT,((SandboxAST) sAssignNode).getLine(),0);
        })
}
| #(dAssignNode:SLASH_ASSIGN (lType=constant
{
    if (lType != null)
        reportError("", ErrorCodeTable.LITERAL_ASSIGN, ((SandboxAST) dAssignNode).getLine(), 0);
}
| leftID=identifier rType=expression
{
    if (leftID != null)
        if (!(syntab.getRepresentationRecord(leftID).isConstant())) {
            AbstractRepresentation rep = syntab.getRepresentation(leftID);

            if (checkBinaryOp (SLASH_ASSIGN, rep.getType(), rType, ((SandboxAST) dAssignNode).getLine(), 0) != null)
                type = checkAssignment (SLASH_ASSIGN, rep.getType(), rType, ((SandboxAST) dAssignNode).getLine(), 0);
            } else
                reportError(leftID + " is a constant; it cannot be modified.",
                    ErrorCodeTable.ASSIGNMENT_TO_CONSTANT,((SandboxAST) dAssignNode).getLine(),0);
        })
}
| #(pcAssignNode:PERCENT_ASSIGN (lType=constant
{
    if (lType != null)
        reportError("", ErrorCodeTable.LITERAL_ASSIGN, ((SandboxAST) pcAssignNode).getLine(), 0);
}
| leftID=identifier rType=expression
{
    if (leftID != null)
        if (!(syntab.getRepresentationRecord(leftID).isConstant())) {
            AbstractRepresentation rep = syntab.getRepresentation(leftID);

            if (checkBinaryOp (PERCENT_ASSIGN, rep.getType(), rType, ((SandboxAST) pcAssignNode).getLine(), 0) != null)
                type = checkAssignment (PERCENT_ASSIGN, rep.getType(), rType, ((SandboxAST) pcAssignNode).getLine(), 0);
            } else
                reportError(leftID + " is a constant; it cannot be modified.",
                    ErrorCodeTable.ASSIGNMENT_TO_CONSTANT,((SandboxAST) pcAssignNode).getLine(),0);
        })
}
| #(eqOpNode:EQ lType=expression rType=expression)
{
    // check the special case of stream equality
    // The "C"-language says that the streams can only be compared to boolean
    if (lType != null && rType != null)
        if ((lType.isStream() && (rType.isBoolType() || rType.isBoolLiteralType()))
            || (rType.isStream() && (lType.isBoolType() || lType.isBoolLiteralType()))))
            type = new BoolType();
        else
            type = checkBinaryOp(EQ, lType, rType, ((SandboxAST) eqOpNode).getLine(), 0);
        else
            type = checkBinaryOp(EQ, lType, rType, ((SandboxAST) eqOpNode).getLine(), 0);
}
| #(neOpNode:NE lType=expression rType=expression)

```

```

{
  // check the special case of stream equality
  // The "C"-language says that the streams can only be compared to boolean
  if (lType != null && rType != null)
    if ((lType.isStream() && (rType.isBoolType() || rType.isBoolLiteralType()))
        || (rType.isStream() && (lType.isBoolType() || (lType.isBoolLiteralType()))))
      type = new BoolType();
    else
      type = checkBinaryOp(NE, lType, rType, ((SandboxAST) neOpNode).getLine(), 0);
  else
    type = checkBinaryOp(NE, lType, rType, ((SandboxAST) neOpNode).getLine(), 0);
}
| #(ltOpNode:LT lType=expression rType=expression)
{
  type = checkBinaryOp(LT, lType, rType, ((SandboxAST) ltOpNode).getLine(), 0);
}
| #(gtOpNode:GT lType=expression rType=expression)
{
  type = checkBinaryOp(GT, lType, rType, ((SandboxAST) gtOpNode).getLine(), 0);
}
| #(leOpNode:LE lType=expression rType=expression)
{
  type = checkBinaryOp(LE, lType, rType, ((SandboxAST) leOpNode).getLine(), 0);
}
| #(geOpNode:GE lType=expression rType=expression)
{
  type = checkBinaryOp(GE, lType, rType, ((SandboxAST) geOpNode).getLine(), 0);
}
| #(postInc:POST_INCR lType=expression)
{
  if (lType != null && lType.isStringType()) {
    reportError("You can not increment this data type.", ErrorCodeTable.INVALID_INCREMENT,
              ((SandboxAST)postInc).getLine(), 0);
  }
  else {
    if (lType != null) {
      java.lang.String id = identifier(postInc.getFirstChild());

      if ((id != null) && (symtab.getRepresentationRecord(id).isConstant()))
        reportError("You can not increment a constant.", ErrorCodeTable.CONST_INCREMENT,
                  ((SandboxAST)postInc).getLine(), 0);
      else
        type = lType;
    }
  }
}
| #(postDec:POST_DECR lType=expression)
{
  if (lType != null && lType.isStringType())
    reportError("You can not decrement this data type.", ErrorCodeTable.INVALID_DECREMENT,
              ((SandboxAST)postDec).getLine(), 0);
  else {
    if (lType != null) {
      java.lang.String id = identifier(postDec.getFirstChild());

      if ((id != null) && (symtab.getRepresentationRecord(id).isConstant()))
        reportError("You can not decrement a constant.", ErrorCodeTable.CONST_DECREMENT,
                  ((SandboxAST)postDec).getLine(), 0);
      else
        type = lType;
    }
  }
}
| #(inc:INCR lType=expression)
{
  if (lType != null && lType.isStringType())
    reportError("You can not increment this data type.", ErrorCodeTable.INVALID_INCREMENT,
              ((SandboxAST)inc).getLine(), 0);
  else {
    if (lType != null) {
      name = (SandboxAST) inc.getFirstChild();
      name2 = (SandboxAST) name.getFirstChild();

      AST postOp = inc.getFirstChild();

```



```

        type = checkBinaryOp(STAR, lType, rType, ((SandboxAST) starOpNode).getLine(), 0);
    }
| #{slashOpNode:SLASH lType=expression rType=expression}
    {
        type = checkBinaryOp(SLASH, lType, rType, ((SandboxAST) slashOpNode).getLine(), 0);
    }
| #{percOpNode:PERCENT lType=expression rType=expression}
    {
        type = checkBinaryOp(PERCENT, lType, rType, ((SandboxAST) percOpNode).getLine(), 0);
    }
| #{landOpNode:LAND lType=expression rType=expression}
    {
        if (lType != null && rType != null)
            if ((lType.isStream() && (rType.getName().equals("bool") || rType.getName().equals("boolliteral")))
                || (rType.isStream() && (lType.getName().equals("bool") || lType.getName().equals("boolliteral"))))
                type = new BoolType();
            else
                type = checkBinaryBooleanOp(LAND, lType, rType, ((SandboxAST) landOpNode).getLine(), 0);
        else
            type = new BoolType();
    }
| #{lorOpNode:LOR lType=expression rType=expression}
    {
        if (lType != null && rType != null)
            if ((lType.isStream() && (rType.getName().equals("bool") || rType.getName().equals("boolliteral")))
                || (rType.isStream() && (lType.getName().equals("bool") || lType.getName().equals("boolliteral"))))
                type = new BoolType();
            else
                type = checkBinaryBooleanOp(LOR, lType, rType, ((SandboxAST) lorOpNode).getLine(), 0);
        else
            type = new BoolType();
    }
| #{bangOpNode:BANG lType=expression}
    {
        type = checkUnaryBooleanOp(BANG, lType, ((SandboxAST) bangOpNode).getLine(), 0);
    }
| #{lshiftOpNode:LSHIFT lType=expression rType=expression}
    {
        try {
            if (lType != null) {
                if (lType.getName().equals("ostream") || lType.getName().equals("ofstream")) {
                    // Enforce 'using namespace std' when using cout or output file streams!
                    if (!usingStdSet && !usingErrMsgShown) {
                        reportError("Namespace 'std' not in use.", ErrorCodeTable.STD_NOT_IN_USE,
                                    ((SandboxAST) lshiftOpNode).getLine(), 0);
                        usingErrMsgShown = true;
                    }
                }

                // Handle screwup seen by student: intValue << inFile;
                if (lType != null && rType != null) {
                    String lTypeName = lType.getName();
                    String rTypeName = rType.getName();

                    if (rTypeName.equals("ifstream") || rTypeName.equals("istream") || rTypeName.equals("ofstream")
                        || rTypeName.equals("ostream"))
                        throw new InvalidOperationException(LSHIFT, rType);
                }
            }
        } catch (InvalidOperationException ioe) {
            reportError(lType.getName() + " " + rType.getName(), ErrorCodeTable.INVALID_OPERATION,
                        ((SandboxAST) lshiftOpNode).getLine(), 0);
            debugStream.println(" Operation between " + lType.getName() + " and " +
                                rType.getName() + " not supported");
        }

        type = checkBinaryOp(LSHIFT, lType, rType, ((SandboxAST) lshiftOpNode).getLine(), 0);
    }
| #{rshiftOpNode:RSHIFT lType=expression rType=expression}
    {
        if (lType != null && rType != null) {
            String lTypeName = lType.getName();
            String rTypeName = rType.getName();

```

```

try {
    if (lTypeName.equals("istream") || lType.equals("ifstream")) {
        // Enforce 'using namespace std' when using cin or input file streams!
        if (!usingStdSet && !usingErrMsgShown) {
            reportError("Namespace 'std' not in use.", ErrorCodeTable.STD_NOT_IN_USE,
                ((SandboxAST) rshiftOpNode).getLine(), 0);
            usingErrMsgShown = true;
        }
    }

    // Ensure that the shift operator is not being used on an invalid type. This check is
    // included because the istream libraries had to register the >> operation on these types
    // in order to work.
    if (lTypeName.equals("bool") || lTypeName.equals("float") || lTypeName.equals("double") ||
        lTypeName.equals("string") || lTypeName.equals("stringliteral"))
        throw new InvalidOperationException(RSHIFT, lType);

    // Ensure that if the left side of the shift operator is a stream (ifstream/istream) then
    // the target for the reading is not a literal.
    if ((lTypeName.equals("istream") || lTypeName.equals("ifstream")) &&
        (rTypeName.equals("charliteral") || rTypeName.equals("intliteral") ||
         rTypeName.equals("stringliteral") || rTypeName.equals("boolliteral") ||
         rTypeName.equals("doubleliteral")))
        throw new InvalidOperationException(RSHIFT, lType);

    type = checkBinaryOp(RSHIFT, lType, rType, ((SandboxAST) rshiftOpNode).getLine(), 0);

} catch (InvalidOperationException ioe) {
    reportError(lTypeName + " " + rTypeName, ErrorCodeTable.INVALID_OPERATION,
        ((SandboxAST) rshiftOpNode).getLine(), 0);
    debugStream.println("  Operation between " + lTypeName + " and " + rTypeName + " not supported");
}
}
}
| #CAST typeNode:TYPENAME rType=expression
{
    AbstractRepresentation rep = syntab.getRepresentation(typeNode.getText());

    if (rep != null && rep.isTypeRepresentation()) {
        type = ((TypeRepresentation) rep).getRepresentedType();
        try {
            typeManager.checkConversion(rType, type);
        } catch (IncompatibleTypesException ex) {
            reportError(rType.getName() + " " + type.getName(),
                ErrorCodeTable.UNABLE_TO_CAST, ((SandboxAST) typeNode).getLine(), 0);
            debugStream.println("  " + rType.getName() + " cannot be cast to " + type.getName());
        }
    } else {
        reportError(typeNode.getText() + " is not a valid datatype.", ErrorCodeTable.NOT_TYPE,
            ((SandboxAST) typeNode).getLine(), 0);
        debugStream.println("  " + typeNode.getText() + " is not a valid datatype.");
    }
}
| type=primaryExpr
;

arguments
: ARGUMENTS
;

```

J.6 ANTLR-produced Output File for the *CS1 Sandbox*'s Semantic Validator (Semantic.java)

```
// $ANTLR 2.7.1: "NewSemantic.g" -> "NewSemantic.java"$
package sandbox.compiler.backend;

import java.io.*;
import java.lang.*;
import java.lang.reflect.*;
import java.util.*;
import sandbox.lib.*;
import sandbox.lib.iostream.*;
import sandbox.client.*;
import sandbox.compiler.*;
import sandbox.compiler.type.*;
import sandbox.compiler.symtab.*;
import sandbox.compiler.frontend.*;
import antlr.*;

import antlr.TreeParser;
import antlr.Token;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.ANTLRException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.collections.impl.BitSet;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

/**
 * $Id: semanticSource.tex,v 1.2 2003/07/02 05:13:29 pete Exp $
 *
 * Performs the semantic check on the parsed AST.
 */
public class NewSemantic extends antlr.TreeParser
    implements NewSemanticTokenTypes
{
    // Boolean indicator for the presense of the 'using' declaration -- used to enforce
    // 'using namespace std' for I/O statements (like cout!)
    private boolean usingStdSet = false;
    private boolean usingErrMsgShown = false;

    // Boolean used to validate function returns are present, when needed
    private boolean returnSeen = false;

    // The symbol table
    SymTab symtab = new LinkSymTab();

    // The errors that are observed during the parse phase
    private Vector errorVector = new Vector();
    private Vector typesDeclared = new Vector();

    // The library manager (which contains the type manager)
    private LibraryManager libraryManager = null;
    private TypeManager typeManager = null;

    // The C language features that are allowed/forbidden.
    private Restriction restriction;

    // Output streams for debugging
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    PrintStream debugStream = new PrintStream(out);

    // Vector of cases, used in switchStatement and labeledStatement
    Stack switchLabelStack = new Stack();

```

```

// Retains a reference to the wrapper of the current function being examined.
// This is so we can obtain a reference to its AST when we check the return type.
private FunctionWrapper currentFunction = null;

// The function manager, manages the addition, use and query of user-defined functions
private FunctionManager fm;

/**
 * Sets the library manager to the parameter, also sets the
 * reference to the type manager.
 */
public void setLibraryManager(LibraryManager mgr) {
    libraryManager = mgr;
    typeManager = libraryManager.getTypeManager();
}

/**
 * Constructor which accepts a reference to the function manager.
 */
public NewSemantic (FunctionManager funcMgr) {
    fm = funcMgr;
}

/**
 * Sets a reference to the current restriction object
 */
public void setRestriction(Restriction restr) {
    restriction = restr;
}

/**
 * Returns a vector of each error seen during semantic checking
 */
public Vector getErrorVector() {
    return(errorVector);
}

/**
 * Overridden reportError method, supporting the error reporting via ANTLR
 */
public void reportError (int code, int line, int col) {
    reportError("", code, line, col);
}

/**
 * ReportError method, supporting the error reporting similiar to what ANTLR does.
 * We added support for the line number and column via this method.
 */
public void reportError (java.lang.String errMsg, int code, int line, int col) {
    AntlrError e = new AntlrError ();
    e.setType("semantic check error");
    e.setErrorMessage(errMsg);
    e.setErrorCode(code);
    e.setLineNumber(line);
    e.setColumnNumber(col);
    errorVector.add(e);
}

/**
 * Overridden reportError method, supporting the error reporting via ANTLR
 */
public void reportError(RecognitionException ex) {
    AntlrError error = new AntlrError();
    System.err.println("Recognition Exception in semantic checker: " + ex.getMessage() + ", line:" + ex.getLine());
    error.setType("recognition exception - semantic checker");
    error.setErrorMessage(ex.getMessage());
    error.setLineNumber(ex.getLine());
}

```

```

        error.setColumnNumber(ex.getColumn());
        errorVector.add(error);
        ex.printStackTrace();
    }

    /**
     * Overridden reportError method, supporting the error reporting via ANTLR.
     * There is no support for the line number or column number via this method.
     */
    public void reportError(java.lang.String s) {
        AntlrError error = new AntlrError();
        error.setType("error - semantic checker");
        error.setErrorMessage(s);
        errorVector.add(error);
    }

    /**
     * Overridden reportError method, supporting the error reporting via ANTLR
     */
    public void reportWarning(java.lang.String s) {
        AntlrError error = new AntlrError();
        error.setType("warning");
        error.setErrorMessage(s);
        errorVector.add(error);
    }

    /**
     * Sets the debugging stream
     */
    public void setDebugOutput(PrintStream p) {
        debugStream = p;
    }

    /**
     * Returns the string representation of the debugging output, for display in
     * debugging window.
     */
    public java.lang.String getDebugOutput() {
        return(out.toString());
    }

    /**
     * Returns the String representation of the identifier 'identifName's' type, assuming it exists.
     */
    private String getType(String identifName) {
        String returnTypeName = null;

        if ((symtab != null) && (identifName != null)) {
            AbstractRepresentation abRep = symtab.getRepresentation(identifName);
            if (abRep != null)
                returnTypeName = abRep.getType().getName();
        }
        return returnTypeName;
    }

    /**
     * Validates the assignment of the specified types. Line number and
     * column number are included for use in the error output (if needed). Returns the
     * left hand type's type as the resulting type of the expression.
     */
    AbstractType checkAssignment(int op, AbstractType lType, AbstractType rType, int line, int col) {
        boolean valid = false;

        if (rType != null && lType != null) {
            debugStream.println("    Checking result of assignment of " + rType.getName() +
                " to " + lType.getName());

            try {

```

```

    valid = typeManager.checkConversion(lType, rType);
  } catch (IncompatibleTypesException ex) {
    valid = false;
  }
}

if (!valid && rType != null && lType != null) {
  reportError(lType.getName() + " " + rType.getName(), ErrorCodeTable.INCOMPATIBLE_TYPES, line, col);
  debugStream.println(" " + lType.getName() + " and " + rType.getName() +
    " are incompatible data types.");
}
debugStream.println("    Assignment results in " + lType.getName() + " type");
} else {
  debugStream.print("    Insufficient type information: (");
  if (lType != null)
    debugStream.print("LHS: " + lType);
  else
    debugStream.print("LHS: null");

  if (rType != null)
    debugStream.print("    RHS: " + rType);
  else
    debugStream.print("    RHS: null");
  debugStream.println(")");
}
return lType;
}

/**
 * Validates the use of the binary operation on the specified types. Line number and
 * column number are included for use in the error output (if needed). Returns the
 * type of value returned from the expression.
 */
AbstractType checkBinaryOp(int op, AbstractType lType, AbstractType rType, int line, int col) {
  AbstractType type = null;

  if (rType != null && lType != null)
    debugStream.println("    Checking result of operation between " + lType.getName() + " and " +
      rType.getName());
  else {
    debugStream.print("    Insufficient type information: (");
    if (lType != null)
      debugStream.print("LHS: " + lType);
    else
      debugStream.print("LHS: null");

    if (rType != null)
      debugStream.print("    RHS: " + rType);
    else
      debugStream.print("    RHS: null");
    debugStream.println(")");
  }

  try {
    type = typeManager.checkBinaryOp(op, lType, rType);
  } catch (IncompatibleTypesException ite) {
    reportError(lType.getName() + " " + rType.getName(), ErrorCodeTable.INCOMPATIBLE_TYPES, line, col);
    debugStream.println(" " + lType.getName() + " and " + rType.getName() +
      " are incompatible data types.");
  } catch (InvalidOperationException ioe) {
    reportError(lType.getName() + " " + rType.getName(), ErrorCodeTable.INVALID_OPERATION, line, col);
    debugStream.println("    Operation between " + lType.getName() + " and " +
      rType.getName() + " not supported");
  }
}

if (type != null) {
  // Return "proper" type for logical operations! Previously this code assumed binary ops to be arithmetic,
  // and did not consider the proper type to return for the logical operators!
  switch (op) {
    case EQ:
    case NE:
    case LAND:
    case LOR:
    case LT:
    case GT:

```

```

        case LE:
        case GE:
            type = new BoolType();
            break;
    }
    debugStream.println("    Result of operation is of type " + type.getName());
}
return type;
}

/**
 * Validates the use of the binary boolean operation on the specified types. Line number and
 * column number are included for use in the error output (if needed). Returns a bool type
 * indicating the type of value returned from the expression.
 */
AbstractType checkBinaryBooleanOp(int op, AbstractType lType, AbstractType rType, int line, int col) {
    AbstractType type = new BoolType();

    if (rType != null && lType != null)
        debugStream.println("    Checking result of boolean operation between " + lType.getName() + " and " +
            rType.getName());
    else {
        debugStream.print("    Insufficient type information: (");
        if (lType != null)
            debugStream.print("LHS: " + lType);
        else
            debugStream.print("LHS: null");

        if (rType != null)
            debugStream.print("    RHS: " + rType);
        else
            debugStream.print("    RHS: null");
        debugStream.println(")");
    }
    try { // try and make the conversion
        typeManager.checkConversion(lType, type);
        typeManager.checkConversion(rType, type);
    } catch (IncompatibleTypesException ex) {
        reportError(lType.getName() + " " + rType.getName(), ErrorCodeTable.INCOMPATIBLE_TYPES, line, col);
        debugStream.println("    " + lType.getName() + " and " + rType.getName() +
            " are incompatible data types.");
    }
    return type;
}

/**
 * Validates the use of the unary boolean operation on the specified type. Line number and
 * column number are included for use in the error output (if needed). Returns a bool type
 * indicating the type of value returned from the expression.
 */
AbstractType checkUnaryBooleanOp(int op, AbstractType lType, int line, int col) {
    AbstractType type = new BoolType();

    if (lType != null)
        debugStream.println("    Checking result of boolean operation on " + lType.getName());
    else
        debugStream.println("    Insufficient type information: (" + lType + ")");

    if (lType.isStream()) // the type is a stream type
        return type;
    try { // try to perform a conversion to bool
        typeManager.checkConversion(lType, type);
    } catch (IncompatibleTypesException ex) {
        reportError(lType.getName(), ErrorCodeTable.INCOMPATIBLE_UNARY_BOOLEAN, line, col);
        debugStream.println("    A " + lType.getName() + " is an incompatible data type for a boolean operation.");
    }
    return type;
}

/**
 * Validates that the variable (value) for the function can be evaluated and is of the correct type.

```

```

* The constantsAllowed parameter allows the function to check if the value is a constant, because
* sometimes the variable cannot be a constant (I.E. - when the variable is being used as a return
* value.
*/
public boolean validVariable(SandboxAST value, String type, boolean constantsAllowed) {
    try {
        // try to obtain the identifier's type
        String typeName = getType(value.getText());
        AbstractType variableType = typeManager.getType(typeName);

        if (variableType == null) {
            try {
                if (!constantsAllowed) // the variable can not be a constant
                    return false;
                // check if the value is a constant
                variableType = constant(value);
                if (variableType == null) // the value is not an identifier or a constant
                    return false;
                // else -- OK
            } catch (RecognitionException re) {
                //I dont know what this does but it isnt good
                return false;
            }
        }

        if (!typeManager.checkConversion(variableType, typeManager.getType(type))) // incompatible type check
            return false;

        return true;
    } catch (IncompatibleTypesException ite) { // incompatible types
        return false;
    }
}

/**
 * Builds the type manager's type map and locates/stores the shortest path from each type to
 * each type. Repeatedly prints out "declaring type" each time it is executed, which is generally
 * more than once (once for primitive types, once for included types).
 */
void assessTypes() {
    typeManager.buildTypeMap();
    for (Enumeration e = typeManager.getTypeNames(); e.hasMoreElements(); ) {
        java.lang.String typeName = (java.lang.String)e.nextElement();
        AbstractType type = typeManager.getType(typeName);

        if (!typesDeclared.contains(typeName))
        {
            typesDeclared.addElement(typeName);
            debugStream.println("Declaring type " + typeName);
        }

        TypeRepresentation rep = new TypeRepresentation(type);
        syntab.setRepresentationRecord(typeName, new RepresentationRecord());
        syntab.setRepresentation(typeName, rep);
    }
}

/**
 * Check to see if the test (used on if, for, while, and do statements) is semantically valid
 */
public boolean validTestExpression(AbstractType type) {
    if (type == null)
        return false;

    return (type.isNumeric() || type.isStream());
}

/**
 * Returns an array of class objects of actual parameter types to a library function call.
 * Actuals is assumed to be the ARGUMENTS node in the AST, so we first need to skip down
 * to the first parameter.

```

```

*/
protected Class[] getActualTypes(AST actuals) {
    if(actuals == null)
        return new Class[] {};

    AST param = (SandboxAST) actuals.getFirstChild();
    ArrayList tempList = new ArrayList();

    while (param != null) {
        try {
            AbstractType rep = expression(param);
            // Handles the base types
            if (rep == null) {
                reportError(param.getText() + " has not been declared." , ErrorCodeTable.NOT_DECLARED,
                    ((SandboxAST) actuals).getLine(), 0);
                debugStream.println(" " + param.getText() + " not declared.");
            }
            else if (rep.isDoubleType() || rep.isDoubleLiteralType() || rep.isFloatType())
                tempList.add(DoubleRepresentation.class);

            else if (rep.isIntType() || rep.isIntLiteralType() || rep.isShortType() || rep.isLongType())
                tempList.add(IntRepresentation.class);

            else if (rep.isBoolType() || rep.isBoolLiteralType())
                tempList.add(BoolRepresentation.class);

            else if (rep.isCharType() || rep.isCharLiteralType())
                tempList.add(CharRepresentation.class);

            else if (rep.getName().equals("istream") || rep.getName().equals("ifstream"))
                tempList.add(IStreamRepresentation.class);

            else if (rep.getName().equals("string"))
                tempList.add(sandbox.lib.string.StringRepresentation.class);

            else if (rep.getName().equals("stringliteral"))
                tempList.add(StringLiteralRepresentation.class);

            else
                debugStream.println(" unknown actual type: " + rep.getName());

            param = param.getNextSibling();
        } catch (RecognitionException re) {
            re.printStackTrace();
        }
    }

    // Build and return the list of class objects
    int count = tempList.size();
    Class[] classArray = new Class[count];
    for (int i=0; i < count; i++)
        classArray[i] = (Class) tempList.get(i);
    return classArray;
}

/**
 * Converts the src type (a Java class, as in Integer.class) to the corresponding
 * internal primitive type (IntType) for use in calculating type conversions when
 * looking up library function calls.
 */
public static AbstractType convertClassToAbstract(Class src) {
    if (src.getName().equals("java.lang.Integer") || src.getName().equals("int"))
        return (new IntType());
    else if (src.getName().equals("java.lang.Double"))
        return new DoubleType();
    else if (src.getName().equals("java.lang.Character"))
        return new CharType();
    else if (src.getName().equals("java.lang.Boolean"))
        return new BoolType();
    else if (src.getName().equals("sandbox.compiler.type.DoubleRepresentation"))
        return new DoubleType();
    else if (src.getName().equals("sandbox.compiler.type.IntRepresentation"))
        return new IntType();
}

```

```

    else
        return null; // this should never occur!
    }
}
public NewSemantic() {
    tokenNames = _tokenNames;
}

public final void program(AST _t) throws RecognitionException {

    AST program_AST_in = (AST)_t;
    AST prog = null;

    debugStream.println("-----");
    debugStream.println("Checking program:");
    assessTypes();

    try { // for error handling
        AST __t2 = _t;
        prog = _t==ASTNULL ? null :(AST)_t;
        match(_t,PROGRAM);
        _t = _t.getFirstChild();
        {
        _loop4:
        do {
            if (_t==null) _t=ASTNULL;
            if ((_t.getType()==INCLUDE)) {
                include(_t);
                _t = _retTree;
            }
            else {
                break _loop4;
            }
        }
        } while (true);
        }
        {
        _loop6:
        do {
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType()) {
            case DECLARATION:
            {
                declaration(_t);
                _t = _retTree;
                break;
            }
            case FUNCTION:
            {
                function(_t);
                _t = _retTree;
                break;
            }
            case FUNCTION_PROTOTYPE:
            {
                functionPrototype(_t);
                _t = _retTree;
                break;
            }
            default:
            {
                break _loop6;
            }
            }
        } while (true);
        }
        _t = __t2;
        _t = _t.getNextSibling();

        /**
         * AFTER checking all the declarations and functions,
         * we finally check if there is a main function.
         */
        if (fm.locateFunction("main", 0) == null) {

```

```

        reportError("No main function. Cannot proceed.",
            ErrorCodeTable.NO_MAIN, 0, 0);
        debugStream.println("    No main function. Cannot proceed.");
    }

    debugStream.println("-----");
    debugStream.println("Function hashtable contents:");
    debugStream.println(fm.toString());
}

}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
}_retTree = _t;
}

public final void include(AST _t) throws RecognitionException {

    AST include_AST_in = (AST)_t;
    AST includeNode = null;

    try {        // for error handling
        includeNode = (AST)_t;
        match(_t, INCLUDE);
        _t = _t.getNextSibling();

        AST name = includeNode.getFirstChild();

        // Since lexer already checks this, we can presume correctness
        try {

            if (name != null) {
                String incName = name.getText().trim();
                if (!restriction.isEnabled(Restriction.CLIMITS) && incName.equals("climits"))
                    reportError ("climits", ErrorCodeTable.HEADER_DISABLED, ((SandboxAST) name).getLine(), 0);
                else if (!restriction.isEnabled(Restriction.CMATH) && incName.equals("cmath"))
                    reportError ("cmath", ErrorCodeTable.HEADER_DISABLED, ((SandboxAST) name).getLine(), 0);
                else if (!restriction.isEnabled(Restriction.CSTDLIB) && incName.equals("cstdlib"))
                    reportError ("cstdlib", ErrorCodeTable.HEADER_DISABLED, ((SandboxAST) name).getLine(), 0);
                else if (!restriction.isEnabled(Restriction.IOMANIP) && incName.equals("iomanip"))
                    reportError ("iomanip", ErrorCodeTable.HEADER_DISABLED, ((SandboxAST) name).getLine(), 0);
                else if (!restriction.isEnabled(Restriction.IOSTREAM) && incName.equals("iostream"))
                    reportError ("iostream", ErrorCodeTable.HEADER_DISABLED, ((SandboxAST) name).getLine(), 0);
                else if (!restriction.isEnabled(Restriction.FSTREAM) && incName.equals("fstream"))
                    reportError ("fstream", ErrorCodeTable.HEADER_DISABLED, ((SandboxAST) name).getLine(), 0);
                else if (!restriction.isEnabled(Restriction.STRING) && incName.equals("string"))
                    reportError ("string", ErrorCodeTable.HEADER_DISABLED, ((SandboxAST) name).getLine(), 0);
                else {
                    Library lib = libraryManager.loadLibrary(incName);
                    debugStream.println("-----");
                    debugStream.println("Checking includes:");
                    debugStream.println("    Loaded library " + incName);
                    assessTypes();
                    if (incName.equals("iostream"))
                        lib.loadSymbols(symtab, null);
                    else
                        lib.loadSymbols(symtab);
                }
            }
        } catch (Exception ex) {
            reportError(ex.toString(), 0, 0, 0);
            debugStream.println("    Error loading library " + name.getText().trim());
            ex.printStackTrace(debugStream);
        }
    }

}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
}_retTree = _t;
}
}

```

```

public final void declaration(AST _t) throws RecognitionException {

    AST declaration_AST_in = (AST)_t;
    AST declNode = null;

    try {        // for error handling
        declNode = (AST)_t;
        match(_t,DECLARATION);
        _t = _t.getNextSibling();

        AST modifierNode = declNode.getFirstChild();
        AST typeNode;

        if (modifierNode.getText().equals("USING_DIRECTIVE")) {
            // Reads the name of the namespace as the "typeNode"
            // Just a hack so that we can recognize the 'using' line and
            // mark that that the "correct" namespace (std) is in use
            typeNode = modifierNode.getFirstChild();
            if (typeNode.getText().equals("std"))
                usingStdSet = true;
        } else {
            if (modifierNode.getText().equals("TYPE_MODIFIER"))
                typeNode = modifierNode.getNextSibling();
            else {
                typeNode = modifierNode;
                modifierNode = null;
            }
        }

        AST declaratorNode = typeNode.getNextSibling();
        SandboxAST mNode = (SandboxAST) typeNode;
        if (declaratorNode == null)
            reportError("Declarator is missing :", ErrorCodeTable.MISSING_DECLARATOR, mNode.getLine(), 0);
        else {
            String dataType = typeNode.getText();

            // Check restrictions on the primitive types!
            if (!restriction.isEnabled(Restriction.BOOLEAN) && dataType.equals("bool"))
                reportError ("boolean", ErrorCodeTable.IDENTIFIER_DISABLED, mNode.getLine(), 0);
            else if (!restriction.isEnabled(Restriction.CHAR) && dataType.equals("char"))
                reportError ("character", ErrorCodeTable.IDENTIFIER_DISABLED, mNode.getLine(), 0);
            else if (!restriction.isEnabled(Restriction.SHORT) && dataType.equals("short"))
                reportError ("short", ErrorCodeTable.IDENTIFIER_DISABLED, mNode.getLine(), 0);
            else if (!restriction.isEnabled(Restriction.INT) && dataType.equals("int"))
                reportError ("integer", ErrorCodeTable.IDENTIFIER_DISABLED, mNode.getLine(), 0);
            else if (!restriction.isEnabled(Restriction.LONG) && dataType.equals("long"))
                reportError ("long", ErrorCodeTable.IDENTIFIER_DISABLED, mNode.getLine(), 0);
            else if (!restriction.isEnabled(Restriction.FLOAT) && dataType.equals("float"))
                reportError ("float", ErrorCodeTable.IDENTIFIER_DISABLED, mNode.getLine(), 0);
            else if (!restriction.isEnabled(Restriction.DOUBLE) && dataType.equals("double"))
                reportError ("double", ErrorCodeTable.IDENTIFIER_DISABLED, mNode.getLine(), 0);

            // Ensure namespace 'std' is in use for strings, ifstream, and ofstream, if applicable.
            if (dataType.equals("string") || dataType.equals("ifstream") || dataType.equals("ofstream"))
                // Enforce 'using namespace std' when using cout or file streams!
                if (!usingStdSet && !usingErrMsgShown) {
                    reportError("Namespace 'std' not in use.", ErrorCodeTable.STD_NOT_IN_USE,
                        ((SandboxAST) typeNode).getLine(), 0);
                    usingErrMsgShown = true;
                }
        }

        AbstractRepresentation rep = symtab.getRepresentation(dataType);
        if ((rep != null) && (rep.isTypeRepresentation()) && (!typeNode.getText().equals("void"))) {
            while (declaratorNode != null) {
                declarator(declaratorNode, typeNode.getText(), modifierNode);
                declaratorNode = declaratorNode.getNextSibling();
            }
        } else {
            reportError(dataType + " is not a valid data type.",
                ErrorCodeTable.NOT_TYPE, mNode.getLine(), 0);
            debugStream.println(dataType + " is not a valid data type.");
        }
    }
}

```

```

    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final void function(AST _t) throws RecognitionException {

    AST function_AST_in = (AST)_t;
    AST funcNode = null;

    try { // for error handling
        funcNode = (AST)_t;
        match(_t,FUNCTION);
        _t = _t.getNextSibling();

        returnSeen = false;

        AST type = funcNode.getFirstChild();
        AST name = type.getNextSibling();
        AST tempNode = name.getNextSibling();

        debugStream.println("-----");
        debugStream.println("Checking function definition: " + name.getText());

        AST paramsNode = null;
        AST block = null;

        if (tempNode.getText().equals("PARAMETER_LIST")) {
            paramsNode = tempNode;
            block = paramsNode.getNextSibling();
        } else if (tempNode.getText().equals("BLOCK_STATEMENT"))
            block = tempNode;

        // Handle case of a function declaration, when we see a block_statement
        if (block != null) {
            currentFunction = fm.addFunction(funcNode, false);
            symtab.beginLevel();

            debugStream.println("  Checking formal parameters:");
            FunctionWrapper tempWrapper = new FunctionWrapper(funcNode, false);

            // Process the formal parameter list
            if (tempWrapper.getParameterListRoot() != null) {
                ArrayList paramPassMechList = new ArrayList();
                AbstractType[] formals = formalParameters(tempWrapper.getParameterListRoot(), true,
                    paramPassMechList, ((SandboxAST) type).getLine());

                // Iterate through paramPassMechList and existing passMechList (from a prototype,
                // if any) and present errors, if any
                boolean foundError = false;
                if (paramPassMechList != null && !paramPassMechList.isEmpty()) {
                    for (int index = 0; index < paramPassMechList.size(); index++) {
                        int functionMech = ((Integer) paramPassMechList.get(index)).intValue();
                        int protoMech = currentFunction.getFormParamPassMech(index);
                        if (protoMech != FunctionWrapper.INVALID && protoMech != functionMech) {
                            reportError(tempWrapper.getParamName(index), ErrorCodeTable.PASSING_MISMATCH,
                                ((SandboxAST) type).getLine(), 0);
                            debugStream.println("Function parameter '" + tempWrapper.getParamName(index) +
                                "' does not match its prototype passing mechanism.");
                            foundError = true;
                        }
                    }
                }
            }
            if (!foundError)
                currentFunction.setFormalParams(formals, paramPassMechList);
        }

        // Check the return type (does it match the prototype, if a proto exists?)
        if (currentFunction.getPrototypeSeen()) {
            AbstractRepresentation rep = symtab.getRepresentation(type.getText());

```

```

        if (rep != null && rep.isTypeRepresentation())
            if (!type.getText().equals(currentFunction.getReturnType())) {
                reportError(type.getText() + " " + currentFunction.getReturnType(),
                    ErrorCodeTable.FUNCTION_TYPE_MISMATCH,
                    ((SandboxAST) type).getLine(), 0);
                debugStream.println("Function return type (" + type.getText() +
                    ") does not match prototype return type (" + currentFunction.getReturnType() +
                    ").");
            }
    }

    debugStream.println("  Checking function body");
    blockWithoutNewAR(block);
    syntab.endLevel();

    // Process the return type (if present)
    debugStream.println("  Validating presence of return statement, if needed.");
    if (!returnSeen && !type.getText().equals("void")) {
        reportError("Function " + name.getText() + " returns a " + type.getText() + " type.",
            ErrorCodeTable.MISSING_RETURN,
            ((SandboxAST) type).getLine(), 0);
        debugStream.println("Function " + name.getText() + " fails to return a " +
            type.getText() + " type.");
    }

    debugStream.println("End of function check.");
}

}

catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
}

public final void functionPrototype(AST _t) throws RecognitionException {

    AST functionPrototype_AST_in = (AST)_t;
    AST pNode = null;

    try {        // for error handling
        pNode = (AST)_t;
        match(_t,FUNCTION_PROTOTYPE);
        _t = _t.getNextSibling();

        AST type = pNode.getFirstChild();
        AST name = type.getNextSibling();

        debugStream.println("-----");
        debugStream.println("Checking function prototype: " + name.getText());
        currentFunction = fm.addFunction(pNode, true);
        if (currentFunction.getParameterListRoot() != null) {
            ArrayList paramPassMechList = new ArrayList();
            AbstractType[] formals = formalParameters(currentFunction.getParameterListRoot(),
                false, paramPassMechList, ((SandboxAST) type).getLine());
            currentFunction.setFormalParams(formals, paramPassMechList);
        }

    }

    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final AbstractType[] formalParameters(AST _t,
    boolean isFunction, ArrayList paramPassMech, int lineNum
) throws RecognitionException {
    AbstractType[] formalParams;

    AST formalParameters_AST_in = (AST)_t;
    AST parameterListNode = null;

```

```

    // The 'isFunction' parameter indicates whether (true) or not (false) we are processing
    // a function definition or a function prototype.

    formalParams = null;

try {    // for error handling
    parameterListNode = (AST)_t;
    match(_t,PARAMETER_LIST);
    _t = _t.getNextSibling();

    Iterator iter;
    int index;
    ArrayList formalParamsList = new ArrayList();

    for (AST node = parameterListNode.getFirstChild(); node != null; node = node.getNextSibling()) {
        formalParamsList.add(formalParameter(node, isFunction, lineNum));

        AST passMechanism = null;
        boolean isConst = false;

        if (node.getFirstChild().getText().equals("TYPE_MODIFIER")) {
            if (node.getFirstChild().getFirstChild().getText().equals("const"))
                isConst = true;
            passMechanism = node.getFirstChild().getNextSibling().getNextSibling();
        } else
            passMechanism = node.getFirstChild().getNextSibling();

        if (passMechanism != null && passMechanism.getText().equals("&"))
            if (!isConst)
                paramPassMech.add(new Integer(FunctionWrapper.BY_REF));
            else
                paramPassMech.add(new Integer(FunctionWrapper.BY_CONST_REF));
        else
            if (!isConst)
                paramPassMech.add(new Integer(FunctionWrapper.BY_VALUE));
            else
                paramPassMech.add(new Integer(FunctionWrapper.BY_CONST_VALUE));
        }
        formalParams = new AbstractType[formalParamsList.size()];
        for (index = 0, iter = formalParamsList.iterator(); iter.hasNext();
            formalParams[index++] = (AbstractType) iter.next();
        )
    }
} catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return formalParams;
}

public final AbstractType formalParameter(AST _t,
    boolean isFunction, int lineNum
) throws RecognitionException {
    AbstractType type;

    AST formalParameter_AST_in = (AST)_t;
    AST formalNode = null;

    // The 'isFunction' parameter indicates whether or not we are processing
    // a function definition (true) or a function prototype (false).

    type = null;

try {    // for error handling
    formalNode = (AST)_t;
    match(_t,PARAMETER);
    _t = _t.getNextSibling();

    AST typeNode = formalNode.getFirstChild();
    boolean isConst = false;
    boolean passedByReference = false;

```

```

// Handle the processing of type modifiers (const)
if (typeNode.getText().equals("TYPE_MODIFIER")) {
    if (typeNode.getFirstChild().getText().equals("const"))
        isConst = true;
    typeNode = typeNode.getNextSibling();
}
debugStream.println("    parameter located: " + typeNode.getText());

// Attempt to obtain the parameter's name
AST paramNode = typeNode.getNextSibling();

// If this is a reference parameter, skip the token and pick up the parameter's name
if (paramNode != null && paramNode.getText().equals("&")) {
    passedByReference = true;
    paramNode = paramNode.getNextSibling();
}

// Check if the parameter exists already in the symbol table.
if (paramNode != null && symtab.isLocal(paramNode.getText())) {
    reportError(paramNode + " already declared!", ErrorCodeTable.REDECLARED, lineNum, 0);
    debugStream.println(paramNode + " already declared!");
} else {
    AbstractRepresentation rep = symtab.getRepresentation(typeNode.getText());

    if (rep != null && rep.isTypeRepresentation()) {
        type = ((TypeRepresentation) rep).getRepresentedType();
        RepresentationRecord newRec = new RepresentationRecord();
        if (isConst)
            newRec.setConstant(true);

        // Ensure streams are passed by reference!
        if (type.isStream() && !passedByReference) {
            reportError(ErrorCodeTable.STREAMS_NOT_PASSED_BY_REF, lineNum, 0);
            debugStream.println("    A file stream parameter was not passed by reference.");
        }

        // Only add function definition formal parameters to the symbol table!
        if (paramNode != null) {
            if (isFunction) {
                symtab.setRepresentationRecord(paramNode.getText(), newRec);
                symtab.setRepresentation(paramNode.getText(), type.instantiate());
            }
            else if (isFunction) {
                reportError(ErrorCodeTable.MISSING_IDENTIFIER_IN_PARAM_LIST, lineNum, 0);
                debugStream.println("    An identifier is missing from the parameter list.");
            }
        }
        else {
            SandboxAST fNode = (SandboxAST) formalNode;
            reportError(typeNode.getText() + " is not a valid datatype.",
                ErrorCodeTable.NOT_TYPE, fNode.getLine(), 0);
            debugStream.println(typeNode.getText() + " is not a valid datatype.");
        }
    }
}

}

} catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return type;
}

/**
 * STATEMENTS PART
 */
public final void statement(AST _t) throws RecognitionException {

    AST statement_AST_in = (AST)_t;

    try {
        // for error handling
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType() ) {
        case WHILE:

```

```
    {
        whileStatement(_t);
        _t = _retTree;
        break;
    }
    case FOR:
    {
        forStatement(_t);
        _t = _retTree;
        break;
    }
    case DO:
    {
        doStatement(_t);
        _t = _retTree;
        break;
    }
    case IF:
    {
        ifStatement(_t);
        _t = _retTree;
        break;
    }
    case SWITCH:
    {
        switchStatement(_t);
        _t = _retTree;
        break;
    }
    case EXPRESSION_STATEMENT:
    {
        expressionStatement(_t);
        _t = _retTree;
        break;
    }
    case BLOCK_STATEMENT:
    {
        blockStatement(_t);
        _t = _retTree;
        break;
    }
    case DECLARATION:
    {
        declaration(_t);
        _t = _retTree;
        break;
    }
    case RETURN:
    {
        returnStatement(_t);
        _t = _retTree;
        break;
    }
    case BREAK:
    {
        breakStatement(_t);
        _t = _retTree;
        break;
    }
    case CONTINUE:
    {
        continueStatement(_t);
        _t = _retTree;
        break;
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
}
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
```

```

    }
    _retTree = _t;
}

public final void whileStatement(AST _t) throws RecognitionException {

    AST whileStatement_AST_in = (AST)_t;
    AST whileNode = null;

    AbstractType type=null;

    try {        // for error handling
        AST __t58 = _t;
        whileNode = _t==ASTNULL ? null :(AST)_t;
        match(_t,WHILE);
        _t = _t.getFirstChild();
        type=expression(_t);
        _t = _retTree;
        {
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType() ) {
            case BREAK:
            case CONTINUE:
            case DO:
            case FOR:
            case IF:
            case RETURN:
            case SWITCH:
            case WHILE:
            case BLOCK_STATEMENT:
            case DECLARATION:
            case EXPRESSION_STATEMENT:
            {
                statement(_t);
                _t = _retTree;
                break;
            }
            case 3:
            {
                break;
            }
            default:
            {
                throw new NoViableAltException(_t);
            }
            }
        }
        _t = __t58;
        _t = _t.getNextSibling();

        if (!validTestExpression(type)) {
            reportError("", ErrorCodeTable.INVALID_BOOLEAN_EXPRESSION, ((SandboxAST)whileNode).getLine(), 0);
        }

    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final void forStatement(AST _t) throws RecognitionException {

    AST forStatement_AST_in = (AST)_t;
    AST forNode = null;

    AbstractType initNode=null;
    AbstractType condNode=null;
    AbstractType iterNode=null;

    try {        // for error handling

```

```

AST _t41 = _t;
forNode = _t==ASTNULL ? null :(AST)_t;
match(_t,FOR);
_t = _t.getFirstChild();
{
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case FOR_INIT:
{
initNode=forInit(_t);
_t = _retTree;
break;
}
case DECLARATION:
{
declaration(_t);
_t = _retTree;
break;
}
default:
{
throw new NoViableAltException(_t);
}
}
}
condNode=forCond(_t);
_t = _retTree;
iterNode=forIter(_t);
_t = _retTree;
{
if (_t==null) _t=ASTNULL;
switch ( _t.getType()) {
case BREAK:
case CONTINUE:
case DO:
case FOR:
case IF:
case RETURN:
case SWITCH:
case WHILE:
case BLOCK_STATEMENT:
case DECLARATION:
case EXPRESSION_STATEMENT:
{
statement(_t);
_t = _retTree;
break;
}
case 3:
{
break;
}
default:
{
throw new NoViableAltException(_t);
}
}
}
_t = _t41;
_t = _t.getNextSibling();

if (condNode != null && !validTestExpression(condNode))
reportError("", ErrorCodeTable.INVALID_BOOLEAN_EXPRESSION, ((SandboxAST) forNode).getLine(), 0);
}
catch (RecognitionException ex) {
reportError(ex);
if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
}

```

```

public final void doStatement(AST _t) throws RecognitionException {

    AST doStatement_AST_in = (AST)_t;
    AST doNode = null;

    AbstractType type=null;

    try {        // for error handling
        AST __t61 = _t;
        doNode = _t==ASTNULL ? null :(AST)_t;
        match(_t,DO);
        _t = _t.getFirstChild();
        blockStatement(_t);
        _t = _retTree;
        type=expression(_t);
        _t = _retTree;
        _t = __t61;
        _t = _t.getNextSibling();

        if (!validTestExpression(type)) {
            reportError("", ErrorCodeTable.INVALID_BOOLEAN_EXPRESSION, ((SandboxAST)doNode).getLine(), 0);
        }

    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final void ifStatement(AST _t) throws RecognitionException {

    AST ifStatement_AST_in = (AST)_t;
    AST ifNode = null;

    AbstractType type=null;

    try {        // for error handling
        AST __t55 = _t;
        ifNode = _t==ASTNULL ? null :(AST)_t;
        match(_t,IF);
        _t = _t.getFirstChild();
        type=expression(_t);
        _t = _retTree;
        statement(_t);
        _t = _retTree;
        {
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType() ) {
            case BREAK:
            case CONTINUE:
            case DO:
            case FOR:
            case IF:
            case RETURN:
            case SWITCH:
            case WHILE:
            case BLOCK_STATEMENT:
            case DECLARATION:
            case EXPRESSION_STATEMENT:
            {
                statement(_t);
                _t = _retTree;
                break;
            }
            case 3:
            {
                break;
            }
            default:
            {

```

```

        throw new NoViableAltException(_t);
    }
}
}
_t = _t55;
_t = _t.getNextSibling();

    if (!validTestExpression(type)) {
        reportError("", ErrorCodeTable.INVALID_BOOLEAN_EXPRESSION, ((SandboxAST)ifNode).getLine(), 0);
    }
}
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
}

public final void switchStatement(AST _t) throws RecognitionException {

    AST switchStatement_AST_in = (AST)_t;
    AST switchNode = null;
    AST exprNode = null;
    AST iNode = null;
    AST cNode = null;
    AST fNode = null;
    AST sNode = null;
    AST tNode = null;
    AST flNode = null;

    debugStream.println("Checking switch statement");

    try {        // for error handling
        AST __t14 = _t;
        switchNode = _t==ASTNULL ? null :(AST)_t;
        match(_t,SWITCH);
        _t = _t.getFirstChild();
        {
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType()) {
            case IDENTIFIER:
            {
                exprNode = (AST)_t;
                match(_t,IDENTIFIER);
                _t = _t.getNextSibling();

                // add a new ArrayList to the switchLabelStack (for the switch statement currently being checked)
                switchLabelStack.push(new ArrayList());

                // Validate that the switch expression is a char, int, charLiteral, or intLiteral
                java.lang.String identifType = getType(exprNode.getText());

                if (identifType == null)
                    reportError(exprNode.getText(), ErrorCodeTable.NOT_DECLARED, ((SandboxAST) exprNode).getLine(), 0);

                else if (!identifType.equals("int") && !identifType.equals("char"))
                    reportError(exprNode.getText() + " is not integer/character; cannot be a switch variable.",
                        ErrorCodeTable.INVALID_TYPE_SWITCH_IDENTIFIER, ((SandboxAST) switchNode).getLine(), 0);

                AST caseGroupNode = exprNode.getNextSibling();
                while (caseGroupNode != null) {
                    casesGroup(caseGroupNode);
                    caseGroupNode = caseGroupNode.getNextSibling();
                }

                // Remove the current ArrayList (of case labels), since we're done processing this switch statement
                switchLabelStack.pop();

                break;
            }
            case STAR:
            case PLUS:

```

```

case MINUS:
case SLASH:
case PERCENT:
{
  {
    if (_t==null) _t=ASTNULL;
    switch ( _t.getType() ) {
    case PLUS:
    {
      AST tmp1_AST_in = (AST)_t;
      match(_t,PLUS);
      _t = _t.getNextSibling();
      break;
    }
    case MINUS:
    {
      AST tmp2_AST_in = (AST)_t;
      match(_t,MINUS);
      _t = _t.getNextSibling();
      break;
    }
    case STAR:
    {
      AST tmp3_AST_in = (AST)_t;
      match(_t,STAR);
      _t = _t.getNextSibling();
      break;
    }
    case SLASH:
    {
      AST tmp4_AST_in = (AST)_t;
      match(_t,SLASH);
      _t = _t.getNextSibling();
      break;
    }
    case PERCENT:
    {
      AST tmp5_AST_in = (AST)_t;
      match(_t,PERCENT);
      _t = _t.getNextSibling();
      break;
    }
    default:
    {
      throw new NoViableAltException(_t);
    }
  }
}

// This is a basic expression, let's let it pass and let the interp handle it.
// These may produce non-int(long) results,
// while may give the interpreter fits, so something here needs to be added to ensure the int result.
AbstractType exprType = expression(switchNode.getFirstChild());
if (!(exprType.isIntType()))
  reportError("", ErrorCodeTable.SWITCH_EXPRESSION_NOT_INT_RESULT,
    ((SandboxAST) switchNode).getLine(), 0);

break;
}
case INTEGER_CONSTANT:
{
  iNode = (AST)_t;
  match(_t,INTEGER_CONSTANT);
  _t = _t.getNextSibling();
  break;
}
case CHAR_CONSTANT:
{
  cNode = (AST)_t;
  match(_t,CHAR_CONSTANT);
  _t = _t.getNextSibling();
  break;
}
case FLOATING_CONSTANT:

```

```

    {
        fNode = (AST)_t;
        match(_t,FLOATING_CONSTANT);
        _t = _t.getNextSibling();
        break;
    }
    case STRING_LITERAL:
    {
        sNode = (AST)_t;
        match(_t,STRING_LITERAL);
        _t = _t.getNextSibling();
        break;
    }
    case TRUE:
    {
        tNode = (AST)_t;
        match(_t,TRUE);
        _t = _t.getNextSibling();
        break;
    }
    case FALSE:
    {
        flNode = (AST)_t;
        match(_t,FALSE);
        _t = _t.getNextSibling();
        break;
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
}
}
_t = _t14;
_t = _t.getNextSibling();

// add a new ArrayList to the switchLabelStack (for the switch statement currently being checked)
switchLabelStack.push(new ArrayList());
AST caseGroupNode = null;

if (iNode != null) {
    reportError(iNode.getText() + " is not integer/character; cannot be a switch variable.",
        ErrorCodeTable.INVALID_TYPE_SWITCH_IDENTIFIER, ((SandboxAST) switchNode).getLine(), 0);
    caseGroupNode = iNode.getNextSibling();
} else if (cNode != null) {
    reportError(cNode.getText() + " is not integer/character; cannot be a switch variable.",
        ErrorCodeTable.INVALID_TYPE_SWITCH_IDENTIFIER, ((SandboxAST) switchNode).getLine(), 0);
    caseGroupNode = cNode.getNextSibling();
} else if (fNode != null) {
    reportError(fNode.getText() + " is not integer/character; cannot be a switch variable.",
        ErrorCodeTable.INVALID_TYPE_SWITCH_IDENTIFIER, ((SandboxAST) switchNode).getLine(), 0);
    caseGroupNode = fNode.getNextSibling();
} else if (sNode != null) {
    reportError(sNode.getText() + " is not integer/character; cannot be a switch variable.",
        ErrorCodeTable.INVALID_TYPE_SWITCH_IDENTIFIER, ((SandboxAST) switchNode).getLine(), 0);
    caseGroupNode = sNode.getNextSibling();
} else if (tNode != null) {
    reportError(tNode.getText() + " is not integer/character; cannot be a switch variable.",
        ErrorCodeTable.INVALID_TYPE_SWITCH_IDENTIFIER, ((SandboxAST) switchNode).getLine(), 0);
    caseGroupNode = tNode.getNextSibling();
} else if (flNode != null) {
    reportError(flNode.getText() + " is not integer/character; cannot be a switch variable.",
        ErrorCodeTable.INVALID_TYPE_SWITCH_IDENTIFIER, ((SandboxAST) switchNode).getLine(), 0);
    caseGroupNode = flNode.getNextSibling();
}

while (caseGroupNode != null) {
    casesGroup(caseGroupNode);
    caseGroupNode = caseGroupNode.getNextSibling();
}

// Remove the current ArrayList (of case labels), since we're done processing this switch statement
switchLabelStack.pop();

```

```

    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final void expressionStatement(AST _t) throws RecognitionException {

    AST expressionStatement_AST_in = (AST)_t;

    AbstractType type=null;

    try { // for error handling
        AST __t63 = _t;
        AST tmp6_AST_in = (AST)_t;
        match(_t,EXPRESSION_STATEMENT);
        _t = _t.getFirstChild();
        type=expression(_t);
        _t = _retTree;
        _t = __t63;
        _t = _t.getNextSibling();
    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final void blockStatement(AST _t) throws RecognitionException {

    AST blockStatement_AST_in = (AST)_t;

    symtab.beginLevel();

    try { // for error handling
        AST __t33 = _t;
        AST tmp7_AST_in = (AST)_t;
        match(_t,BLOCK_STATEMENT);
        _t = _t.getFirstChild();
        {
        _loop35:
        do {
            if (_t==null) _t=ASTNULL;
            if ((_tokenSet_0.member(_t.getType())) {
                statement(_t);
                _t = _retTree;
            }
            else {
                break _loop35;
            }
        } while (true);
        }
        _t = __t33;
        _t = _t.getNextSibling();

        symtab.endLevel();

    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final void returnStatement(AST _t) throws RecognitionException {

    AST returnStatement_AST_in = (AST)_t;

```

```

AST returnNode = null;

AbstractType type=null;

try { // for error handling
  AST __t28 = _t;
  returnNode = _t==ASTNULL ? null :(AST)_t;
  match(_t,RETURN);
  _t = _t.getFirstChild();
  {
  if (_t==null) _t=ASTNULL;
  switch ( _t.getType() ) {
  case FLOATING_CONSTANT:
  case FALSE:
  case TRUE:
  case LPAREN:
  case INCR:
  case DECR:
  case STAR:
  case PLUS:
  case MINUS:
  case BANG:
  case SLASH:
  case PERCENT:
  case LSHIFT:
  case RSHIFT:
  case LT:
  case GT:
  case LE:
  case GE:
  case EQ:
  case NE:
  case LAND:
  case LOR:
  case ASSIGN:
  case STAR_ASSIGN:
  case SLASH_ASSIGN:
  case PERCENT_ASSIGN:
  case PLUS_ASSIGN:
  case MINUS_ASSIGN:
  case COMMA:
  case IDENTIFIER:
  case INTEGER_CONSTANT:
  case CHAR_CONSTANT:
  case STRING_LITERAL:
  case CALL:
  case CAST:
  case POST_DECR:
  case POST_INCR:
  {
    type=expression(_t);
    _t = _retTree;
    break;
  }
  case 3:
  {
    break;
  }
  default:
  {
    throw new NoViableAltException(_t);
  }
  }
  }
  _t = __t28;
  _t = _t.getNextSibling();

  debugStream.println(" Checking the return type (" + currentFunction.getName() + ")");
  debugStream.println(" Return type is: " + currentFunction.getReturnType());

  try {
    returnSeen = true;
    AbstractType typeToReturn = ((TypeRepresentation)

```

```

        symtab.getRepresentation(currentFunction.getReturnType()).getRepresentedType();
        if (!typeToReturn.isVoidType() && type == null) {
            reportError("null " + currentFunction.getReturnType(),
                ErrorCodeTable.RETURN_TYPE_MISMATCH, ((SandboxAST) returnNode).getLine(), 0);
            debugStream.println("    " + currentFunction.getReturnType() +
                " and *null* are incompatible data types");
        } else
            typeManager.checkConversion(typeToReturn, type);
    }
    catch (IncompatibleTypesException ite) {
        reportError(type.getName() + " " + currentFunction.getReturnType(),
            ErrorCodeTable.RETURN_TYPE_MISMATCH, ((SandboxAST) returnNode).getLine(), 0);
        debugStream.println("    " + currentFunction.getReturnType() + " and " + type.getName() +
            " are incompatible data types");
    }
}
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
}
_retTree = _t;
}

public final void breakStatement(AST _t) throws RecognitionException {

    AST breakStatement_AST_in = (AST)_t;

    try {        // for error handling
        AST tmp8_AST_in = (AST)_t;
        match(_t,BREAK);
        _t = _t.getNextSibling();
    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final void continueStatement(AST _t) throws RecognitionException {

    AST continueStatement_AST_in = (AST)_t;

    try {        // for error handling
        AST tmp9_AST_in = (AST)_t;
        match(_t,CONTINUE);
        _t = _t.getNextSibling();
    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final void casesGroup(AST _t) throws RecognitionException {

    AST casesGroup_AST_in = (AST)_t;

    AST sList = null;

    try {        // for error handling
        AST __t18 = _t;
        AST tmp10_AST_in = (AST)_t;
        match(_t,CASE_GROUP);
        _t = _t.getFirstChild();
        {
            int _cnt20=0;
            _loop20:
            do {
                if (_t==null) _t=ASTNULL;
                if ((_t.getType()==CASE||_t.getType()==DEFAULT)) {

```

```

        acase(_t);
        _t = _retTree;
    }
    else {
        if ( _cnt20>=1 ) { break _loop20; } else {throw new NoViableAltException(_t);}
    }

    _cnt20++;
} while (true);
}
sList=caseSList(_t);
_t = _retTree;
_t = _t18;
_t = _t.getNextSibling();
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
}

public final void acase(AST _t) throws RecognitionException {

    AST acase_AST_in = (AST)_t;
    AST caseNode = null;
    AST defNode = null;

    AbstractType type;
    ArrayList currentLabelList = (ArrayList) switchLabelStack.peek();

    try {        // for error handling
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType() ) {
        case CASE:
        {
            AST __t22 = _t;
            caseNode = _t==ASTNULL ? null :(AST)_t;
            match(_t,CASE);
            _t = _t.getFirstChild();
            type=expression(_t);
            _t = _retTree;
            _t = __t22;
            _t = _t.getNextSibling();

            // Check to see if the label is charLiteral or intLiteral
            AST caseLabel = caseNode.getFirstChild();
            if (currentLabelList.contains(caseLabel.getText()))
                reportError(caseLabel.getText(), ErrorCodeTable.DUPLICATED_CASE, ((SandboxAST) caseNode).getLine(), 0);
            else {
                if (getType(caseLabel.getText()) == null) {
                    // The type of the constant (literal) is returned here
                    AbstractType labelType = constant(caseLabel);

                    // If the constant is not an integer or character an error is thrown
                    if ((labelType != null) && ((labelType.isIntLiteralType() || (labelType.isCharLiteralType()))
                    currentLabelList.add(caseLabel.getText());
                } else
                    reportError(caseLabel.getText(), ErrorCodeTable.INVALID_CASE,
                    ((SandboxAST) caseLabel).getLine(), 0);
            } else {
                // Case label had a type so it's likely an identifier... but what kind?
                String ident = identifier(caseLabel);
                if ((ident != null) && (symtab.getRepresentationRecord(ident).isConstant())) {
                    if (!getType(ident).equals("int") && !getType(ident).equals("char"))
                        reportError(ident, ErrorCodeTable.INVALID_CASE, ((SandboxAST) caseNode).getLine(), 0);
                    else
                        currentLabelList.add(caseLabel.getText());
                } else
                    reportError(caseNode.getText(), ErrorCodeTable.INVALID_CASE, ((SandboxAST) caseNode).getLine(), 0);
            }
        }
        }
    }
}

```

```

        break;
    }
    case DEFAULT:
    {
        defNode = (AST)_t;
        match(_t,DEFAULT);
        _t = _t.getNextSibling();

        // We can have only one default in any switch statement. The first one seen is added
        // to the currentLabelList. If encountered again, then an error is thrown.
        if (!currentLabelList.contains("default"))
            currentLabelList.add("default");
        else
            reportError("", ErrorCodeTable.TOO_MANY_SWITCH_DEFAULTS, ((SandboxAST) defNode).getLine(), 0);

        break;
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
}
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
}

public final AST caseSList(AST _t) throws RecognitionException {
    AST firstStatement;

    AST caseSList_AST_in = (AST)_t;
    AST sNode = null;

    firstStatement = null;

    try {
        // for error handling
        AST _t24 = _t;
        sNode = _t==ASTNULL ? null :(AST)_t;
        match(_t,SLIST);
        _t = _t.getFirstChild();
        {
            int _cnt26=0;
            _loop26:
            do {
                if (_t==null) _t=ASTNULL;
                if ((_tokenSet_0.member(_t.getType())) {
                    statement(_t);
                    _t = _retTree;
                }
                else {
                    if ( _cnt26>=1 ) { break _loop26; } else {throw new NoViableAltException(_t);}
                }
            }
            _cnt26++;
        } while (true);
    }
    _t = _t24;
    _t = _t.getNextSibling();

    // Nothing to check here, just let them trickle into the statement checker
    if (firstStatement == null)
        firstStatement = sNode.getFirstChild();

}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return firstStatement;

```

```

}

public final AbstractType expression(AST _t) throws RecognitionException {
    AbstractType type;

    AST expression_AST_in = (AST)_t;
    AST callNode = null;
    AST commaNode = null;
    AST assignNode = null;
    AST pAssignNode = null;
    AST mAssignNode = null;
    AST sAssignNode = null;
    AST dAssignNode = null;
    AST pcAssignNode = null;
    AST eqOpNode = null;
    AST neOpNode = null;
    AST ltOpNode = null;
    AST gtOpNode = null;
    AST leOpNode = null;
    AST geOpNode = null;
    AST postInc = null;
    AST postDec = null;
    AST inc = null;
    AST dec = null;
    AST plusOpNode = null;
    AST minusOpNode = null;
    AST starOpNode = null;
    AST slashOpNode = null;
    AST percOpNode = null;
    AST landOpNode = null;
    AST lorOpNode = null;
    AST bangOpNode = null;
    AST lshiftOpNode = null;
    AST rshiftOpNode = null;
    AST typeNode = null;

    SandboxAST name, name2 = null, name3=null, name4=null;
    AbstractType lType=null, rType=null, negType, lType2, lType3, cons;
    java.lang.String leftID;

    if (_t != null)
        debugStream.println("Checking expression: " + _t.toStringList());
    else
        debugStream.println("Checking expression: ***expression is null***");
    type = null;

    try { // for error handling
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType() ) {
        case CALL:
            {
                callNode = (AST)_t;
                match(_t,CALL);
                _t = _t.getNextSibling();

                name = (SandboxAST) callNode.getFirstChild();
                java.lang.String streamName = name.getText();
                java.lang.String typeName = getType(streamName);
                java.lang.String functionName = streamName;

                if (typeName != null && (typeName.equals("ofstream") || typeName.equals("ifstream"))) {
                    // Enforce 'using namespace std' when using file streams!
                    if (!usingStdSet && !usingErrMsgShown) {
                        reportError("Namespace 'std' not in use.", ErrorCodeTable.STD_NOT_IN_USE,
                            ((SandboxAST) name).getLine(), 0);
                        usingErrMsgShown = true;
                    }
                }

                // Reads and ignores the 'dot' in the tree
                name = (SandboxAST) name.getNextSibling();
                // Reads the name of the method call
                name = (SandboxAST) name.getNextSibling();
                //
            }
        }
    }
}

```

```

// OPEN() method call
//
if (name.getText().equals("open")) {
    name2 = (SandboxAST) name.getNextSibling();
    if (name2 == null)
        reportError(name.getText(), ErrorCodeTable.ARGUMENT_NULL, name.getLine(), 0);
    else {
        name2 = (SandboxAST) name2.getFirstChild();
        name3 = (SandboxAST) name2.getNextSibling();
        if (name3 != null)
            reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);

        String argument = name2.getText();
        int invalidCh = '/';
        int invalidChar = '\\';

        if (argument.equals("") || argument.indexOf(invalidCh) != -1 ||
            argument.indexOf(invalidChar) != -1)
            reportError(name.getText(), ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);
        else {
            // Obtain a handle to the representation of the stream and call setOpened method
            if (typeName.equals("ofstream")) {
                OFStreamRepresentation fstreamRep =
                    (OFStreamRepresentation) symtab.getRepresentation(streamName);
                fstreamRep.setOpened(true);
            } else {
                IFStreamRepresentation fstreamRep =
                    (IFStreamRepresentation) symtab.getRepresentation(streamName);
                fstreamRep.setOpened(true);
            }
        }
    }
}
//
// CLOSE() method call
//
} else if (name.getText().equals("close")){
    name2=(SandboxAST) name.getNextSibling();
    if (name2 != null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
    else {
        // Obtain a handle to the representation of the stream and call setOpened method
        if (typeName.equals("ofstream")) {
            OFStreamRepresentation fstreamRep =
                (OFStreamRepresentation) symtab.getRepresentation(streamName);
            fstreamRep.setOpened(false);
        } else {
            IFStreamRepresentation fstreamRep =
                (IFStreamRepresentation) symtab.getRepresentation(streamName);
            fstreamRep.setOpened(false);
        }
    }
}
//
// FAIL() method call
//
else if (name.getText().equals("fail") )
{
    name2=(SandboxAST) name.getNextSibling();
    if (name2 != null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);

    type = new BoolType();
}
//
// CLEAR() method call
//
else if (name.getText().equals("clear") )
{
    name2=(SandboxAST) name.getNextSibling();
    if (name2 != null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
}
//
// EOF() method call

```

```

//
else if (name.getText().equals("eof") )
{
    name2=(SandboxAST) name.getNextSibling();
    if (name2 != null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);

    type= new BoolType();
}
//
// PEEK() method call
//
else if (name.getText().equals("peek") ) {
    name2=(SandboxAST) name.getNextSibling();
    if (name2 != null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);

    type= new IntType();
}
//
// IGNORE() method call
//
else if (name.getText().equals("ignore") && typeName.equals("ifstream")) {
    name2 = (SandboxAST) name.getNextSibling();
    if (name2 == null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
    else {
        // Check for parameters of ignore now.
        name2 = (SandboxAST) name2.getFirstChild();
        name3 = (SandboxAST) name2.getNextSibling();

        // Check the first parameter
        AbstractType exprType = expression(name2);
        if (exprType == null || (!(exprType.isIntType()) && !(exprType.isIntLiteralType()))
            reportError(functionName, ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);

        // Check if the second paramter exists, if so then check if it is valid
        if (name3 != null) {
            // Ignore call has 2 parameters

            // Check for a 3rd parameter
            name4 = (SandboxAST) name3.getNextSibling();
            if (name4 != null)
                reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
            else
                // Check 2nd parameter validity
                exprType = expression(name3);
                if (!(exprType.isCharType()) && !(exprType.isCharLiteralType()))
                    reportError(functionName, ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);
        }
    }
}
//
// GET() method call
//
else if (name.getText().equals("get")) {
    name2=(SandboxAST) name.getNextSibling();

    if (name2 == null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
    else {
        // Check for parameter type of get now...should be a character
        name2 = (SandboxAST) name2.getFirstChild();
        name3 = (SandboxAST) name2.getNextSibling();
        if (name3 != null)
            reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
        if (!validVariable(name2, "char", false))
            reportError(name.getText(), ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);
    }
}
//
// PUTBACK() method call
//
else if (name.getText().equals("putback")) {

```

```

    name2=(SandboxAST) name.getNextSibling();

    if (name2 == null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
    else {
        // Check for parameter type of get/putback now...should be a character
        name2 = (SandboxAST) name2.getFirstChild();
        name3 = (SandboxAST) name2.getNextSibling();
        java.lang.String identifType = getType(name2.getText());

        if (name2 == null || name3 != null)
            reportError(name.getText(), ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
        if (!validVariable(name2, "char", true) && !validVariable(name2, "charliteral", true))
            reportError(name.getText(), ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);
    }
}
else {
    reportError(name.getText(), ErrorCodeTable.UNABLE_TO_LOCATE_FUNCTION,
        ((SandboxAST) name).getLine(), 0);
    debugStream.println(" " + name.getText() + " function not found, possible scope issue...");
}
} else if (typeName != null && typeName.equals("istream")) {
    // Reads and ignores the 'dot' in the tree
    name = (SandboxAST) name.getNextSibling();

    // Reads the name of the method call
    name = (SandboxAST) name.getNextSibling();

    // Enforce 'using namespace std' when using input streams!
    if (!usingStdSet && !usingErrMsgShown) {
        reportError("Namespace 'std' not in use.", ErrorCodeTable.STD_NOT_IN_USE,
            ((SandboxAST) name).getLine(), 0);
        usingErrMsgShown = true;
    }
}

if (name.getText().equals("get")) {
    // Obtain the number and type of arguments...
    name2=(SandboxAST) name.getNextSibling();

    if (name2 == null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
    else {
        // check for parameter type of get now...should be a character
        name2 = (SandboxAST) name2.getFirstChild();
        name3 = (SandboxAST) name2.getNextSibling();
        if (name3 != null)
            reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
        if (!validVariable(name2, "char", false))
            reportError(name.getText(), ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);
    }
} else if (name.getText().equals("ignore")) {
    name2=(SandboxAST) name.getNextSibling();
    if (name2 == null)
        reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
    else {
        // Check for parameters of ignore now.
        name2 = (SandboxAST) name2.getFirstChild();
        name3 = (SandboxAST) name2.getNextSibling();

        // Check the first parameter
        AbstractType exprType = expression(name2);
        if (exprType == null || !(exprType.isIntType() && !exprType.isIntLiteralType()))
            reportError(name.getText(), ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);

        // Check if the second paramter exists, if so then check if its valid
        if (name3 != null) {
            // Ignore call has 2 parameters

            // Check for a 3rd parameter
            name4 = (SandboxAST) name3.getNextSibling();
            if (name4 != null)
                reportError(ErrorCodeTable.PARAM_COUNT_MISMATCH, name.getLine(), 0);
            else {
                // Check 2nd parameter validity

```



```

        System.err.println("Exception while obtaining library function:" + name.getText());
        e.printStackTrace();
    }
} else if (functionName.equals("setprecision") && libraryManager.containsFunction("setprecision")) {
    AST argsNode = name.getNextSibling();
    AbstractType exprType = expression(argsNode.getFirstChild());
    if (exprType == null || !(exprType.isShortType()) && !(exprType.isIntType()) &&
        !(exprType.isLongType()) && !(exprType.isIntLiteralType()))
        reportError(functionName, ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);
} else if (functionName.equals("setw") && libraryManager.containsFunction("setw")) {
    AST argsNode = name.getNextSibling();
    AbstractType exprType = expression(argsNode.getFirstChild());
    if (exprType == null || !(exprType.isShortType()) && !(exprType.isIntType()) &&
        !(exprType.isLongType()) && !(exprType.isIntLiteralType()))
        reportError(functionName, ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);
} else if (functionName.equals("setfill") && libraryManager.containsFunction("setfill")) {
    AST argsNode = name.getNextSibling();
    AbstractType exprType = expression(argsNode.getFirstChild());
    if (exprType == null || !exprType.isCharLiteralType()) {
        reportError(functionName, ErrorCodeTable.ARGUMENT_INVALID, name.getLine(), 0);
    }
} else {
    // Handle semantic checks for user-defined functions and library function calls

    String funcName = name.getText();
    int actualParamCount = 0;

    // Count the arguments from the function call, and try to locate the
    // user-defined function via the function manager.
    AST argsNode = name.getNextSibling();
    if (argsNode != null) {
        AST oneArg = argsNode.getFirstChild();
        while (oneArg != null) {
            actualParamCount++;
            oneArg = oneArg.getNextSibling();
        }
    }

    // Attempt to locate the desired function and it's number of parameters
    // If we find it, use it. User-defined functions come before library ones!
    FunctionWrapper function = fm.locateFunction(funcName, actualParamCount);
    if (function != null) {
        function.setCallLineNum(((SandboxAST) name).getLine());
        int formalParamCount = function.getNumOfParams();

        // If the parameter count matches, attempt to validate coercion for each param
        if (formalParamCount == actualParamCount && formalParamCount > 0) {
            AbstractType[] formalParams = function.getFormalParams();
            AbstractType[] actualParams = new AbstractType[actualParamCount];
            AST arg = argsNode.getFirstChild();

            for (int index = 0; index < actualParams.length; index++, arg = arg.getNextSibling())
                actualParams[index] = expression(arg);

            for (int index = 0; index < formalParams.length; index++) {
                if (actualParams[index] != null)
                    try {
                        typeManager.checkConversion(formalParams[index], actualParams[index]);
                    } catch (IncompatibleTypesException ex) {
                        reportError(actualParams[index].getName() + " " + formalParams[index].getName(),
                            ErrorCodeTable.UNABLE_TO_CAST, ((SandboxAST) name).getLine(), 0);
                        debugStream.println(" " + actualParams[index].getName() +
                            " cannot be coerced to " + formalParams[index].getName());
                    }
            }
            type = ((TypeRepresentation)
                symtab.getRepresentation(function.getReturnType())).getRepresentedType();
        }
    } else if (formalParamCount > 0) {
        reportError("Number of parameters do not match. Function '" + funcName +
            "' expects " + formalParamCount + " arguments.",
            ErrorCodeTable.PARAM_COUNT_MISMATCH, ((SandboxAST) name).getLine(), 0);
        debugStream.println("Parameter count mismatch in function '" + funcName + "'.");
    }
}

```

```

} else if (libraryManager.containsFunction(funcName)) {
    // The call is to a library function
    debugStream.println("Calling library function: " + funcName);

    // Ensure what's in the library is call-able, otherwise, it's likely to be
    // handled here in the interpreter, above, as a special function (for I/O)!
    if (libraryManager.inLibrary(funcName)) {
        Class libClass = null;
        Class[] actTypes = getActualTypes(name.getNextSibling());

        try {
            libClass = Class.forName(libraryManager.getOwner(funcName));
            Method libMethod = libClass.getMethod(funcName, actTypes);
            // This is a major hack and a bad way to test the constant modification,
            // but how to we do this inside
            // the library. That is, we can check the situation, but how do we
            // create and report the error from in there?
            // Likely, we'd need to throw an exception and then try to catch it out
            // here... think in the big picture for
            // other situations like this, can we do that for those as well?
            if (funcName.equals("getline")) {
                String secondParam = name.getNextSibling().getFirstChild().getNextSibling().getText();
                if (symtab.getRepresentationRecord(secondParam).isConstant()) {
                    reportError(secondParam, ErrorCodeTable.ASSIGNMENT_TO_CONSTANT,
                        ((SandboxAST) name).getLine(), 0);
                    debugStream.println("    " + secondParam + " is a constant, and can not be modified.");
                }
            }
        }
    }

    } catch (NoSuchMethodException nsme) {
        // OK, we didn't find an exact match for the intended function call,
        // let's check for a close match (look for one with the same name and same
        // number of params (and try to coerce)
        debugStream.println("    Direct function match not found,
            searching for close match...");
        Method[] libMethods = libClass.getMethods();
        debugStream.println("    Checking through " + libMethods.length +
            " methods of " + libClass.getName());
        debugStream.println("    Looking for a match with " +
            actTypes.length + " parameter(s).");
        int bestFitIndex = -1;
        int leastConversions = 999;
        for (int index=0; index < libMethods.length; index++) {
            // Since we get ALL public methods, we need to check
            // both the names and the paramlist size
            if (libMethods[index].getName().equals(funcName) &&
                libMethods[index].getParameterTypes().length == actTypes.length) {

                // Start worrying finding each possible matching function here
                Class[] libParamTypes = libMethods[index].getParameterTypes();
                debugStream.println("    Function match found (based on name): " +
                    libParamTypes.length + " param(s).");
                int sumConversions = 0;
                for (int paramIndex = 0; paramIndex < libParamTypes.length; paramIndex++)
                    sumConversions += typeManager.checkConversionSteps(
                        NewSemantic.convertClassToAbstract(libParamTypes[paramIndex]),
                        NewSemantic.convertClassToAbstract(actTypes[paramIndex]));

                debugStream.println("    The number of conversions for all parameters is: " +
                    sumConversions);
                // Keep a reference to the best one match found
                if (sumConversions > 0 && sumConversions < leastConversions) {
                    leastConversions = sumConversions;
                    bestFitIndex = index;
                }
            }
        }

        if (bestFitIndex != -1) {
            // If we've found matches, set the return type to be what the best
            // fit function returns.
            type = NewSemantic.convertClassToAbstract(libMethods[bestFitIndex].getReturnType());
        }
    }
} else {

```

```

        // If we didn't find a match, give an error
        reportError(name.getText(), ErrorCodeTable.UNABLE_TO_LOCATE_FUNCTION,
            ((SandboxAST) name).getLine(), 0);
        debugStream.println(name.getText() + " function not found, possible scope issue...");
    }

    } catch (Exception e) {
        debugStream.println("...exception caught");
        System.err.println("Exception while obtaining library function:" + funcName);
        e.printStackTrace();
    }
    } else {
        reportError(name.getText(), ErrorCodeTable.UNABLE_TO_LOCATE_FUNCTION,
            ((SandboxAST) name).getLine(), 0);
        debugStream.println(name.getText() + " function not found, possible scope issue...");
    }
    } else {
        reportError(name.getText(), ErrorCodeTable.UNABLE_TO_LOCATE_FUNCTION,
            ((SandboxAST) name).getLine(), 0);
        debugStream.println(name.getText() + " function not found, possible scope issue...");
    }
    }
}

break;
}
case COMMA:
{
    AST _t82 = _t;
    commaNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,COMMA);
    _t = _t.getFirstChild();
    {
        lType=expression(_t);
        _t = _retTree;
        rType=expression(_t);
        _t = _retTree;
    }
    _t = _t82;
    _t = _t.getNextSibling();
    break;
}
case ASSIGN:
{
    AST _t84 = _t;
    assignNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,ASSIGN);
    _t = _t.getFirstChild();
    {
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType() ) {
        case FLOATING_CONSTANT:
        case FALSE:
        case TRUE:
        case INTEGER_CONSTANT:
        case CHAR_CONSTANT:
        case STRING_LITERAL:
        {
            lType=constant(_t);
            _t = _retTree;

            if (lType != null)
                reportError("", ErrorCodeTable.LITERAL_ASSIGN, ((SandboxAST) assignNode).getLine(), 0);

            break;
        }
    }
}
case IDENTIFIER:
{
    leftID=identifier(_t);
    _t = _retTree;
    rType=expression(_t);
    _t = _retTree;
}

```

```

        if (leftID != null)
            if (!syntab.getRepresentationRecord(leftID).isConstant()) {
                AbstractRepresentation rep = syntab.getRepresentation(leftID);

                type = checkAssignment (ASSIGN, rep.getType(), rType,
                    ((SandboxAST) assignNode).getLine(), 0);
            } else
                reportError(leftID + " is a constant; it can not be modified.",
                    ErrorCodeTable.ASSIGNMENT_TO_CONSTANT,
                    ((SandboxAST) assignNode).getLine(), 0);

            break;
        }
    default:
    {
        throw new NoViableAltException(_t);
    }
}
_t = _t84;
_t = _t.getNextSibling();
break;
}
case PLUS_ASSIGN:
{
    AST _t86 = _t;
    pAssignNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,PLUS_ASSIGN);
    _t = _t.getFirstChild();
    {
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType() ) {
        case FLOATING_CONSTANT:
        case FALSE:
        case TRUE:
        case INTEGER_CONSTANT:
        case CHAR_CONSTANT:
        case STRING_LITERAL:
        {
            lType=constant(_t);
            _t = _retTree;

            if (lType != null)
                reportError("", ErrorCodeTable.LITERAL_ASSIGN, ((SandboxAST) pAssignNode).getLine(), 0);

            break;
        }
    }
    case IDENTIFIER:
    {
        leftID=identifier(_t);
        _t = _retTree;
        rType=expression(_t);
        _t = _retTree;

        if (leftID != null)
            if (!syntab.getRepresentationRecord(leftID).isConstant()) {
                AbstractRepresentation rep = syntab.getRepresentation(leftID);

                if (checkBinaryOp (PLUS_ASSIGN, rep.getType(), rType,
                    ((SandboxAST) pAssignNode).getLine(), 0) != null)
                    type = checkAssignment (PLUS_ASSIGN, rep.getType(), rType,
                        ((SandboxAST) pAssignNode).getLine(), 0);
            }
            else
                reportError(leftID + " is a constant; it cannot be modified.",
                    ErrorCodeTable.ASSIGNMENT_TO_CONSTANT,((SandboxAST) pAssignNode).getLine(),0);

            break;
        }
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
}
}

```

```

    }
    _t = _t86;
    _t = _t.getNextSibling();
    break;
}
}
case MINUS_ASSIGN:
{
    AST _t88 = _t;
    mAssignNode = _t==ASTNULL ? null : (AST)_t;
    match(_t, MINUS_ASSIGN);
    _t = _t.getFirstChild();
    {
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType() ) {
        case FLOATING_CONSTANT:
        case FALSE:
        case TRUE:
        case INTEGER_CONSTANT:
        case CHAR_CONSTANT:
        case STRING_LITERAL:
        {
            lType=constant(_t);
            _t = _retTree;

            if (lType != null)
                reportError("", ErrorCodeTable.LITERAL_ASSIGN, ((SandboxAST) mAssignNode).getLine(), 0);

            break;
        }
    }
    case IDENTIFIER:
    {
        leftID=identifier(_t);
        _t = _retTree;
        rType=expression(_t);
        _t = _retTree;

        if (leftID != null)
            if (!(syntab.getRepresentationRecord(leftID).isConstant())) {
                AbstractRepresentation rep = syntab.getRepresentation(leftID);

                if (checkBinaryOp (MINUS_ASSIGN, rep.getType(), rType,
                    ((SandboxAST) mAssignNode).getLine(), 0) != null)
                    type = checkAssignment (MINUS_ASSIGN, rep.getType(), rType,
                        ((SandboxAST) mAssignNode).getLine(), 0);
            } else
                reportError(leftID + " is a constant; it cannot be modified.",
                    ErrorCodeTable.ASSIGNMENT_TO_CONSTANT, ((SandboxAST) mAssignNode).getLine(), 0);

            break;
        }
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
    }
    _t = _t88;
    _t = _t.getNextSibling();
    break;
}
}
case STAR_ASSIGN:
{
    AST _t90 = _t;
    sAssignNode = _t==ASTNULL ? null : (AST)_t;
    match(_t, STAR_ASSIGN);
    _t = _t.getFirstChild();
    {
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType() ) {
        case FLOATING_CONSTANT:
        case FALSE:
        case TRUE:
        case INTEGER_CONSTANT:
        case CHAR_CONSTANT:

```

```

case STRING_LITERAL:
{
  lType=constant(_t);
  _t = _retTree;

  if (lType != null)
    reportError("", ErrorCodeTable.LITERAL_ASSIGN, ((SandboxAST) sAssignNode).getLine(), 0);

  break;
}
case IDENTIFIER:
{
  leftID=identifier(_t);
  _t = _retTree;
  rType=expression(_t);
  _t = _retTree;

  if (leftID != null)
    if (!(syntab.getRepresentationRecord(leftID).isConstant())) {
      AbstractRepresentation rep = syntab.getRepresentation(leftID);

      if (checkBinaryOp (STAR_ASSIGN, rep.getType(), rType,
        ((SandboxAST) sAssignNode).getLine(), 0) != null)
        type = checkAssignment (STAR_ASSIGN, rep.getType(), rType,
          ((SandboxAST) sAssignNode).getLine(), 0);
    } else
      reportError(leftID + " is a constant; it cannot be modified.",
        ErrorCodeTable.ASSIGNMENT_TO_CONSTANT, ((SandboxAST) sAssignNode).getLine(), 0);

  break;
}
default:
{
  throw new NoViableAltException(_t);
}
}
_t = _t90;
_t = _t.getNextSibling();
break;
}
case SLASH_ASSIGN:
{
  AST _t92 = _t;
  dAssignNode = _t==ASTNULL ? null :(AST)_t;
  match(_t, SLASH_ASSIGN);
  _t = _t.getFirstChild();
  {
  if (_t==null) _t=ASTNULL;
  switch ( _t.getType() ) {
  case FLOATING_CONSTANT:
  case FALSE:
  case TRUE:
  case INTEGER_CONSTANT:
  case CHAR_CONSTANT:
  case STRING_LITERAL:
  {
    lType=constant(_t);
    _t = _retTree;

    if (lType != null)
      reportError("", ErrorCodeTable.LITERAL_ASSIGN, ((SandboxAST) dAssignNode).getLine(), 0);

    break;
  }
}
}
case IDENTIFIER:
{
  leftID=identifier(_t);
  _t = _retTree;
  rType=expression(_t);
  _t = _retTree;

  if (leftID != null)
    if (!(syntab.getRepresentationRecord(leftID).isConstant())) {

```

```

        AbstractRepresentation rep = syntab.getRepresentation(leftID);

        if (checkBinaryOp (SLASH_ASSIGN, rep.getType(), rType,
            ((SandboxAST) dAssignNode).getLine(), 0) != null)
            type = checkAssignment (SLASH_ASSIGN, rep.getType(), rType,
                ((SandboxAST) dAssignNode).getLine(), 0);
        } else
            reportError(leftID + " is a constant; it cannot be modified.",
                ErrorCodeTable.ASSIGNMENT_TO_CONSTANT, ((SandboxAST) dAssignNode).getLine(), 0);

        break;
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
    }
    _t = _t92;
    _t = _t.getNextSibling();
    break;
}
case PERCENT_ASSIGN:
{
    AST _t94 = _t;
    pcAssignNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,PERCENT_ASSIGN);
    _t = _t.getFirstChild();
    {
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType() ) {
        case FLOATING_CONSTANT:
        case FALSE:
        case TRUE:
        case INTEGER_CONSTANT:
        case CHAR_CONSTANT:
        case STRING_LITERAL:
        {
            lType=constant(_t);
            _t = _retTree;

            if (lType != null)
                reportError("", ErrorCodeTable.LITERAL_ASSIGN, ((SandboxAST) pcAssignNode).getLine(), 0);

            break;
        }
    }
    case IDENTIFIER:
    {
        leftID=identifier(_t);
        _t = _retTree;
        rType=expression(_t);
        _t = _retTree;

        if (leftID != null)
            if (!(syntab.getRepresentationRecord(leftID).isConstant())) {
                AbstractRepresentation rep = syntab.getRepresentation(leftID);

                if (checkBinaryOp (PERCENT_ASSIGN, rep.getType(), rType,
                    ((SandboxAST) pcAssignNode).getLine(), 0) != null)
                    type = checkAssignment (PERCENT_ASSIGN, rep.getType(), rType,
                        ((SandboxAST) pcAssignNode).getLine(), 0);
                } else
                    reportError(leftID + " is a constant; it cannot be modified.",
                        ErrorCodeTable.ASSIGNMENT_TO_CONSTANT, ((SandboxAST) pcAssignNode).getLine(), 0);

                break;
            }
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
    }
    _t = _t94;

```

```

    _t = _t.getNextSibling();
    break;
}
case EQ:
{
    AST __t96 = _t;
    eqOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,EQ);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    rType=expression(_t);
    _t = _retTree;
    _t = __t96;
    _t = _t.getNextSibling();

    // check the special case of stream equality
    // The "C"-language says that the streams can only be compared to boolean
    if (lType != null && rType != null)
        if ((lType.isStream() && ((rType.isBoolType() || rType.isBoolLiteralType()))
            || (rType.isStream() && ((lType.isBoolType() || (lType.isBoolLiteralType())))))
            type = new BoolType();
        else
            type = checkBinaryOp(EQ, lType, rType, ((SandboxAST) eqOpNode).getLine(), 0);
    else
        type = checkBinaryOp(EQ, lType, rType, ((SandboxAST) eqOpNode).getLine(), 0);

    break;
}
case NE:
{
    AST __t97 = _t;
    neOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,NE);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    rType=expression(_t);
    _t = _retTree;
    _t = __t97;
    _t = _t.getNextSibling();

    // check the special case of stream equality
    // The "C"-language says that the streams can only be compared to boolean
    if (lType != null && rType != null)
        if ((lType.isStream() && ((rType.isBoolType() || rType.isBoolLiteralType()))
            || (rType.isStream() && ((lType.isBoolType() || (lType.isBoolLiteralType())))))
            type = new BoolType();
        else
            type = checkBinaryOp(NE, lType, rType, ((SandboxAST) neOpNode).getLine(), 0);
    else
        type = checkBinaryOp(NE, lType, rType, ((SandboxAST) neOpNode).getLine(), 0);

    break;
}
case LT:
{
    AST __t98 = _t;
    ltOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,LT);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    rType=expression(_t);
    _t = _retTree;
    _t = __t98;
    _t = _t.getNextSibling();

    type = checkBinaryOp(LT, lType, rType, ((SandboxAST) ltOpNode).getLine(), 0);

    break;
}
case GT:
{

```

```

AST __t99 = _t;
gtOpNode = _t==ASTNULL ? null :(AST)_t;
match(_t,GT);
_t = _t.getFirstChild();
lType=expression(_t);
_t = _retTree;
rType=expression(_t);
_t = _retTree;
_t = __t99;
_t = _t.getNextSibling();

        type = checkBinaryOp(GT, lType, rType, ((SandboxAST) gtOpNode).getLine(), 0);

    break;
}
case LE:
{
    AST __t100 = _t;
    leOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,LE);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    rType=expression(_t);
    _t = _retTree;
    _t = __t100;
    _t = _t.getNextSibling();

        type = checkBinaryOp(LE, lType, rType, ((SandboxAST) leOpNode).getLine(), 0);

    break;
}
case GE:
{
    AST __t101 = _t;
    geOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,GE);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    rType=expression(_t);
    _t = _retTree;
    _t = __t101;
    _t = _t.getNextSibling();

        type = checkBinaryOp(GE, lType, rType, ((SandboxAST)geOpNode).getLine(), 0);

    break;
}
case POST_INCR:
{
    AST __t102 = _t;
    postInc = _t==ASTNULL ? null :(AST)_t;
    match(_t,POST_INCR);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    _t = __t102;
    _t = _t.getNextSibling();

        if (lType != null && lType.isStringType()) {
            reportError("You can not increment this data type.", ErrorCodeTable.INVALID_INCREMENT,
                ((SandboxAST)postInc).getLine(), 0);
        }
        else {
            if (lType != null) {
                java.lang.String id = identifier(postInc.getFirstChild());

                if ((id != null) && (symtab.getRepresentationRecord(id).isConstant()))
                    reportError("You can not increment a constant.", ErrorCodeTable.CONST_INCREMENT,
                        ((SandboxAST)postInc).getLine(), 0);
                else
                    type = lType;
            }
        }
}

```

```

        }

        break;
    }
    case POST_DECR:
    {
        AST __t103 = _t;
        postDec = _t==ASTNULL ? null :(AST)_t;
        match(_t,POST_DECR);
        _t = _t.getFirstChild();
        lType=expression(_t);
        _t = _retTree;
        _t = __t103;
        _t = _t.getNextSibling();

        if (lType!= null && lType.isStringType())
            reportError("You can not decrement this data type.", ErrorCodeTable.INVALID_DECREMENT,
                ((SandboxAST)postDec).getLine(), 0);
        else {
            if (lType != null) {
                java.lang.String id = identifier(postDec.getFirstChild());

                if ((id != null) && (symtab.getRepresentationRecord(id).isConstant()))
                    reportError("You can not decrement a constant.", ErrorCodeTable.CONST_DECREMENT,
                        ((SandboxAST)postDec).getLine(), 0);
                else
                    type = lType;
            }
        }

        break;
    }
    case INCR:
    {
        AST __t104 = _t;
        inc = _t==ASTNULL ? null :(AST)_t;
        match(_t,INCR);
        _t = _t.getFirstChild();
        lType=expression(_t);
        _t = _retTree;
        _t = __t104;
        _t = _t.getNextSibling();

        if (lType!= null && lType.isStringType())
            reportError("You can not increment this data type.", ErrorCodeTable.INVALID_INCREMENT,
                ((SandboxAST)inc).getLine(), 0);
        else {
            if (lType != null) {
                name = (SandboxAST) inc.getFirstChild();
                name2= (SandboxAST) name.getFirstChild();

                AST postOp = inc.getFirstChild();

                if (postOp.getType() == POST_INCR) reportError("Invalid use of unary increment operator.",
                    ErrorCodeTable.POST_PRE_INCREMENT, ((SandboxAST) inc).getLine(), 0);
                else {
                    if (postOp.getType() == POST_DECR) reportError("Invalid use of unary decrement operator.",
                        ErrorCodeTable.POST_PRE_INCREMENT, ((SandboxAST) inc).getLine(), 0);
                    else {
                        if (name2 == null) {
                            java.lang.String id = identifier(name);
                            if ((id != null) && (symtab.getRepresentationRecord(id).isConstant()))
                                reportError("You can not increment a constant.", ErrorCodeTable.CONST_INCREMENT,
                                    ((SandboxAST) inc).getLine(), 0);
                            else
                                type = lType;
                        }
                    }
                }
            }
        }

        break;
    }
}

```

```

case DECR:
{
    AST __t105 = _t;
    dec = _t==ASTNULL ? null :(AST)_t;
    match(_t,DECR);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    _t = __t105;
    _t = _t.getNextSibling();

        if (lType!= null && lType.isStringType())
            reportError("You can not decrement this data type.", ErrorCodeTable.INVALID_DECREMENT,
                ((SandboxAST)dec).getLine(), 0);
        else {
            if (lType != null) {
                name = (SandboxAST) dec.getFirstChild();
                name2= (SandboxAST) name.getFirstChild();

                AST postOp = dec.getFirstChild();

                if (postOp.getType() == POST_INCR)
                    reportError("Invalid use of unary increment operator.", ErrorCodeTable.POST_PRE_DECREMENT,
                        ((SandboxAST) dec).getLine(), 0);
                else {
                    if (postOp.getType() == POST_DECR)
                        reportError("Invalid use of unary decrement operator.", ErrorCodeTable.POST_PRE_DECREMENT,
                            ((SandboxAST) dec).getLine(), 0);
                    else {
                        if (name2 == null) {
                            java.lang.String id = identifier(name);

                            if ((id != null) && (symtab.getRepresentationRecord(id).isConstant()))
                                reportError("You can not decrement a constant.", ErrorCodeTable.CONST_DECREMENT,
                                    ((SandboxAST) dec).getLine(), 0);
                            else
                                type = lType;
                        }
                    }
                }
            }
        }

    break;
}
case PLUS:
{
    AST __t106 = _t;
    plusOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,PLUS);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    rType=expression(_t);
    _t = _retTree;
    _t = __t106;
    _t = _t.getNextSibling();

        type = checkBinaryOp(PLUS, lType, rType, ((SandboxAST) plusOpNode).getLine(), 0);

    break;
}
case MINUS:
{
    AST __t107 = _t;
    minusOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,MINUS);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    {
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType() ) {
            case FLOATING_CONSTANT:

```

```

case FALSE:
case TRUE:
case LPAREN:
case INCR:
case DECR:
case STAR:
case PLUS:
case MINUS:
case BANG:
case SLASH:
case PERCENT:
case LSHIFT:
case RSHIFT:
case LT:
case GT:
case LE:
case GE:
case EQ:
case NE:
case LAND:
case LOR:
case ASSIGN:
case STAR_ASSIGN:
case SLASH_ASSIGN:
case PERCENT_ASSIGN:
case PLUS_ASSIGN:
case MINUS_ASSIGN:
case COMMA:
case IDENTIFIER:
case INTEGER_CONSTANT:
case CHAR_CONSTANT:
case STRING_LITERAL:
case CALL:
case CAST:
case POST_DECR:
case POST_INCR:
{
    rType=expression(_t);
    _t = _retTree;
    break;
}
case 3:
{
    break;
}
default:
{
    throw new NoViableAltException(_t);
}
}
}
_t = __t107;
_t = _t.getNextSibling();

if (rType != null)
    type = checkBinaryOp(MINUS, lType, rType, ((SandboxAST) minusOpNode).getLine(), 0);

else if(lType != null) { // rType == null - this is a negation operation not a minus operation
    name = (SandboxAST) minusOpNode.getFirstChild();

    if (!lType.isNumeric()) // test to see that the type is negatable (numeric)
        reportError(name.getText() + " is not a negatable type", ErrorCodeTable.INVALID_NEGATION,
            ((SandboxAST) minusOpNode).getLine(), 0);
    else
        type = expression(name); // Return the type of 'name'
}

break;
}
case STAR:
{
    AST __t109 = _t;
    starOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,STAR);
}

```

```

    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    rType=expression(_t);
    _t = _retTree;
    _t = __t109;
    _t = _t.getNextSibling();

        type = checkBinaryOp(STAR, lType, rType, ((SandboxAST) starOpNode).getLine(), 0);

    break;
}
case SLASH:
{
    AST __t110 = _t;
    slashOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,SLASH);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    rType=expression(_t);
    _t = _retTree;
    _t = __t110;
    _t = _t.getNextSibling();

        type = checkBinaryOp(SLASH, lType, rType, ((SandboxAST) slashOpNode).getLine(), 0);

    break;
}
case PERCENT:
{
    AST __t111 = _t;
    percOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,PERCENT);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    rType=expression(_t);
    _t = _retTree;
    _t = __t111;
    _t = _t.getNextSibling();

        type = checkBinaryOp(PERCENT, lType, rType, ((SandboxAST) percOpNode).getLine(), 0);

    break;
}
case LAND:
{
    AST __t112 = _t;
    landOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,LAND);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    rType=expression(_t);
    _t = _retTree;
    _t = __t112;
    _t = _t.getNextSibling();

        if (lType != null && rType != null)
            if ((lType.isStream() && (rType.getName().equals("bool") || rType.getName().equals("boolliteral")))
                || (rType.isStream() && (lType.getName().equals("bool")
                    || lType.getName().equals("boolliteral"))))
                type = new BoolType();
            else
                type = checkBinaryBooleanOp(LAND, lType, rType, ((SandboxAST) landOpNode).getLine(), 0);
            else
                type = new BoolType();

    break;
}
case LOR:
{
    AST __t113 = _t;

```

```

    lorOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,LOR);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    rType=expression(_t);
    _t = _retTree;
    _t = _t113;
    _t = _t.getNextSibling();

    if (lType != null && rType != null)
        if ((lType.isStream() && (rType.getName().equals("bool") || rType.getName().equals("boolliteral")))
            || (rType.isStream() && (lType.getName().equals("bool")
            || lType.getName().equals("boolliteral"))))
            type = new BoolType();
        else
            type = checkBinaryBooleanOp(LOR, lType, rType, ((SandboxAST) lorOpNode).getLine(), 0);
        else
            type = new BoolType();

    break;
}
case BANG:
{
    AST _t114 = _t;
    bangOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,BANG);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    _t = _t114;
    _t = _t.getNextSibling();

    type = checkUnaryBooleanOp(BANG, lType, ((SandboxAST) bangOpNode).getLine(), 0);

    break;
}
case LSHIFT:
{
    AST _t115 = _t;
    lshiftOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,LSHIFT);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    rType=expression(_t);
    _t = _retTree;
    _t = _t115;
    _t = _t.getNextSibling();

    try {
        if (lType != null) {
            if (lType.getName().equals("ostream") || lType.getName().equals("ofstream")) {
                // Enforce 'using namespace std' when using cout or output file streams!
                if (!usingStdSet && !usingErrMsgShown) {
                    reportError("Namespace 'std' not in use.", ErrorCodeTable.STD_NOT_IN_USE,
                        ((SandboxAST) lshiftOpNode).getLine(), 0);
                    usingErrMsgShown = true;
                }
            }
        }

        // Handle screwup seen by student: intValue << inFile;
        if (lType != null && rType != null) {
            String lTypeName = lType.getName();
            String rTypeName = rType.getName();

            if (rTypeName.equals("ifstream") || rTypeName.equals("istream") ||
                rTypeName.equals("ofstream") || rTypeName.equals("ostream"))
                throw new InvalidOperationException(LSHIFT, rType);
        }
    }
    } catch (InvalidOperationException ioe) {
        reportError(lType.getName() + " " + rType.getName(), ErrorCodeTable.INVALID_OPERATION,
            ((SandboxAST) lshiftOpNode).getLine(), 0);
    }
}

```

```

        debugStream.println("  Operation between " + lType.getName() +
            " and " + rType.getName() + " not supported");
    }

    type = checkBinaryOp(LSHIFT, lType, rType, ((SandboxAST) lshiftOpNode).getLine(), 0);

    break;
}
case RSHIFT:
{
    AST __t116 = _t;
    rshiftOpNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,RSHIFT);
    _t = _t.getFirstChild();
    lType=expression(_t);
    _t = _retTree;
    rType=expression(_t);
    _t = _retTree;
    _t = __t116;
    _t = _t.getNextSibling();

    if (lType != null && rType != null) {
        String lTypeName = lType.getName();
        String rTypeName = rType.getName();

        try {
            if (lTypeName.equals("istream") || lType.equals("ifstream")) {
                // Enforce 'using namespace std' when using cin or input file streams!
                if (!usingStdSet && !usingErrMsgShown) {
                    reportError("Namespace 'std' not in use.", ErrorCodeTable.STD_NOT_IN_USE,
                        ((SandboxAST) rshiftOpNode).getLine(), 0);
                    usingErrMsgShown = true;
                }
            }

            // Ensure that the shift operator is not being used on an invalid type. This check is
            // included because the istream libraries had to register the >> operation on these types
            // in order to work.
            if (lTypeName.equals("bool") || lTypeName.equals("float")
                || lTypeName.equals("double") || lTypeName.equals("string")
                || lTypeName.equals("stringliteral"))
                throw new InvalidOperationException(RSHIFT, lType);

            // Ensure that if the left side of the shift operator is a stream (ifstream/istream) then
            // the target for the reading is not a literal.
            if ((lTypeName.equals("istream") || lTypeName.equals("ifstream")) &&
                (rTypeName.equals("charliteral") || rTypeName.equals("intliteral") ||
                 rTypeName.equals("stringliteral") || rTypeName.equals("boolliteral") ||
                 rTypeName.equals("doubleliteral")))
                throw new InvalidOperationException(RSHIFT, lType);

            type = checkBinaryOp(RSHIFT, lType, rType, ((SandboxAST) rshiftOpNode).getLine(), 0);

        } catch (InvalidOperationException ioe) {
            reportError(lTypeName + " " + rTypeName, ErrorCodeTable.INVALID_OPERATION,
                ((SandboxAST) rshiftOpNode).getLine(), 0);
            debugStream.println("  Operation between " + lTypeName +
                " and " + rTypeName + " not supported");
        }
    }

    break;
}
case CAST:
{
    AST __t117 = _t;
    AST tmp11_AST_in = (AST)_t;
    match(_t,CAST);
    _t = _t.getFirstChild();
    typeNode = (AST)_t;
    match(_t,TYPENAME);
    _t = _t.getNextSibling();
    rType=expression(_t);
    _t = _retTree;

```

```

    _t = _t117;
    _t = _t.getNextSibling();

    AbstractRepresentation rep = sytab.getRepresentation(typeNode.getText());

    if (rep != null && rep.isTypeRepresentation()) {
        type = ((TypeRepresentation) rep).getRepresentedType();
        try {
            typeManager.checkConversion(rType, type);
        } catch (IncompatibleTypesException ex) {
            reportError(rType.getName() + " " + type.getName(),
                ErrorCodeTable.UNABLE_TO_CAST, ((SandboxAST) typeNode).getLine(), 0);
            debugStream.println(" " + rType.getName() + " cannot be cast to " + type.getName());
        }
    } else {
        reportError(typeNode.getText() + " is not a valid datatype.", ErrorCodeTable.NOT_TYPE,
            ((SandboxAST) typeNode).getLine(), 0);
        debugStream.println(" " + typeNode.getText() + " is not a valid datatype.");
    }

    break;
}
case FLOATING_CONSTANT:
case FALSE:
case TRUE:
case LPAREN:
case IDENTIFIER:
case INTEGER_CONSTANT:
case CHAR_CONSTANT:
case STRING_LITERAL:
{
    type=primaryExpr(_t);
    _t = _retTree;
    break;
}
default:
{
    throw new NoViableAltException(_t);
}
}
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return type;
}

public final void blockWithoutNewAR(AST _t) throws RecognitionException {

    AST blockWithoutNewAR_AST_in = (AST)_t;

    try { // for error handling
        AST __t37 = _t;
        AST tmp12_AST_in = (AST)_t;
        match(_t,BLOCK_STATEMENT);
        _t = _t.getFirstChild();
        {
        _loop39:
        do {
            if (_t==null) _t=ASTNULL;
            if ((_tokenSet_0.member(_t.getType())) {
                statement(_t);
                _t = _retTree;
            }
            else {
                break _loop39;
            }
        }
        } while (true);
    }
    _t = __t37;
    _t = _t.getNextSibling();
}

```

```

    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final AbstractType forInit(AST _t) throws RecognitionException {
    AbstractType exprNode;

    AST forInit_AST_in = (AST)_t;
    AST forinit = null;

    exprNode=null;

    try { // for error handling
        AST __t46 = _t;
        forinit = _t==ASTNULL ? null :(AST)_t;
        match(_t,FOR_INIT);
        _t = _t.getFirstChild();
        {
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType()) {
            case FLOATING_CONSTANT:
            case FALSE:
            case TRUE:
            case LPAREN:
            case INCR:
            case DECR:
            case STAR:
            case PLUS:
            case MINUS:
            case BANG:
            case SLASH:
            case PERCENT:
            case LSHIFT:
            case RSHIFT:
            case LT:
            case GT:
            case LE:
            case GE:
            case EQ:
            case NE:
            case LAND:
            case LOR:
            case ASSIGN:
            case STAR_ASSIGN:
            case SLASH_ASSIGN:
            case PERCENT_ASSIGN:
            case PLUS_ASSIGN:
            case MINUS_ASSIGN:
            case COMMA:
            case IDENTIFIER:
            case INTEGER_CONSTANT:
            case CHAR_CONSTANT:
            case STRING_LITERAL:
            case CALL:
            case CAST:
            case POST_DECR:
            case POST_INCR:
            {
                exprNode=expression(_t);
                _t = _retTree;
                break;
            }
            case 3:
            {
                break;
            }
            default:
            {
                throw new NoViableAltException(_t);
            }
            }
        }
    }
}

```

```

    }
    }
    }
    _t = _t46;
    _t = _t.getNextSibling();
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return exprNode;
}

public final AbstractType forCond(AST _t) throws RecognitionException {
    AbstractType exprNode;

    AST forCond_AST_in = (AST)_t;
    AST forcond = null;

    exprNode=null;

    try { // for error handling
        AST _t49 = _t;
        forcond = _t==ASTNULL ? null :(AST)_t;
        match(_t,FOR_COND);
        _t = _t.getFirstChild();
        {
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType() ) {
            case FLOATING_CONSTANT:
            case FALSE:
            case TRUE:
            case LPAREN:
            case INCR:
            case DECR:
            case STAR:
            case PLUS:
            case MINUS:
            case BANG:
            case SLASH:
            case PERCENT:
            case LSHIFT:
            case RSHIFT:
            case LT:
            case GT:
            case LE:
            case GE:
            case EQ:
            case NE:
            case LAND:
            case LOR:
            case ASSIGN:
            case STAR_ASSIGN:
            case SLASH_ASSIGN:
            case PERCENT_ASSIGN:
            case PLUS_ASSIGN:
            case MINUS_ASSIGN:
            case COMMA:
            case IDENTIFIER:
            case INTEGER_CONSTANT:
            case CHAR_CONSTANT:
            case STRING_LITERAL:
            case CALL:
            case CAST:
            case POST_DECR:
            case POST_INCR:
            {
                exprNode=expression(_t);
                _t = _retTree;
                break;
            }
            case 3:

```

```

    {
        break;
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
    }
    _t = _t49;
    _t = _t.getNextSibling();
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return exprNode;
}

public final AbstractType forIter(AST _t) throws RecognitionException {
    AbstractType exprNode;

    AST forIter_AST_in = (AST)_t;
    AST forincr = null;

    exprNode=null;

    try { // for error handling
        AST _t52 = _t;
        forincr = _t==ASTNULL ? null :(AST)_t;
        match(_t,FOR_ITER);
        _t = _t.getFirstChild();
        {
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType() ) {
            case FLOATING_CONSTANT:
            case FALSE:
            case TRUE:
            case LPAREN:
            case INCR:
            case DECR:
            case STAR:
            case PLUS:
            case MINUS:
            case BANG:
            case SLASH:
            case PERCENT:
            case LSHIFT:
            case RSHIFT:
            case LT:
            case GT:
            case LE:
            case GE:
            case EQ:
            case NE:
            case LAND:
            case LOR:
            case ASSIGN:
            case STAR_ASSIGN:
            case SLASH_ASSIGN:
            case PERCENT_ASSIGN:
            case PLUS_ASSIGN:
            case MINUS_ASSIGN:
            case COMMA:
            case IDENTIFIER:
            case INTEGER_CONSTANT:
            case CHAR_CONSTANT:
            case STRING_LITERAL:
            case CALL:
            case CAST:
            case POST_DECR:
            case POST_INCR:

```

```

    {
        exprNode=expression(_t);
        _t = _retTree;
        break;
    }
    case 3:
    {
        break;
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
    }
    _t = _t52;
    _t = _t.getNextSibling();
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return exprNode;
}

public final void declarator(AST _t,
    java.lang.String typeName, AST modNode
) throws RecognitionException {

    AST declarator_AST_in = (AST)_t;
    AST typeN = null;
    AST declNode = null;
    AST id = null;
    AST arrayDecl = null;

    SandboxAST paramsNode= null;
    AbstractType type = null;

    try {        // for error handling
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType() ) {
        case TYPENAME:
        {
            typeN = (AST)_t;
            match(_t,TYPENAME);
            _t = _t.getNextSibling();

            reportError(typeN.getText() + " is an invalid identifier name.", ErrorCodeTable.INVALID_IDENTIFIER_NAME,
                ((SandboxAST) typeN).getLine(), ((SandboxAST) typeN).getLine());

            break;
        }
        case DECLARATOR:
        {
            AST __t66 = _t;
            declNode = _t==ASTNULL ? null :(AST)_t;
            match(_t,DECLARATOR);
            _t = _t.getFirstChild();
            {
                if (_t==null) _t=ASTNULL;
                switch ( _t.getType() ) {
                case IDENTIFIER:
                {
                    id = (AST)_t;
                    match(_t,IDENTIFIER);
                    _t = _t.getNextSibling();
                    {
                        if (_t==null) _t=ASTNULL;
                        switch ( _t.getType() ) {
                        case 3:
                        case FLOATING_CONSTANT:
                        case FALSE:

```

```

case TRUE:
case LPAREN:
case INCR:
case DECR:
case STAR:
case PLUS:
case MINUS:
case BANG:
case SLASH:
case PERCENT:
case LSHIFT:
case RSHIFT:
case LT:
case GT:
case LE:
case GE:
case EQ:
case NE:
case LAND:
case LOR:
case ASSIGN:
case STAR_ASSIGN:
case SLASH_ASSIGN:
case PERCENT_ASSIGN:
case PLUS_ASSIGN:
case MINUS_ASSIGN:
case COMMA:
case IDENTIFIER:
case INTEGER_CONSTANT:
case CHAR_CONSTANT:
case STRING_LITERAL:
case CALL:
case CAST:
case POST_DECR:
case POST_INCR:
{
  {
    if (_t==null) _t=ASTNULL;
    switch ( _t.getType() ) {
    case FLOATING_CONSTANT:
    case FALSE:
    case TRUE:
    case LPAREN:
    case INCR:
    case DECR:
    case STAR:
    case PLUS:
    case MINUS:
    case BANG:
    case SLASH:
    case PERCENT:
    case LSHIFT:
    case RSHIFT:
    case LT:
    case GT:
    case LE:
    case GE:
    case EQ:
    case NE:
    case LAND:
    case LOR:
    case ASSIGN:
    case STAR_ASSIGN:
    case SLASH_ASSIGN:
    case PERCENT_ASSIGN:
    case PLUS_ASSIGN:
    case MINUS_ASSIGN:
    case COMMA:
    case IDENTIFIER:
    case INTEGER_CONSTANT:
    case CHAR_CONSTANT:
    case STRING_LITERAL:
    case CALL:
    case CAST:

```



```

        rep = repType.instantiate();
        symtab.setRepresentation(name, rep);
        if (modNode != null)
            typeModifier(modNode, id.toString());
        if (type != null)
            checkAssignment(ASSIGN, repType, type, iNode.getLine(), 0);
    }
}

break;
}
case ARRAY_DECLARATOR:
{
    arrayDecl = (AST)_t;
    match(_t, ARRAY_DECLARATOR);
    _t = _t.getNextSibling();
    break;
}
default:
{
    throw new NoViableAltException(_t);
}
}
}
_t = _t66;
_t = _t.getNextSibling();

if (arrayDecl != null) {

AST initialNode = arrayDecl.getNextSibling();
AST identifNode = arrayDecl.getFirstChild();
AST sizeNode = identifNode.getNextSibling();

if (symtab.isLocal(identifNode.getText())) {
    reportError(identifNode.getText() + " already declared!", ErrorCodeTable.REDECLARED,
        ((SandboxAST) identifNode).getLine(), 0);
    debugStream.println(identifNode.getText() + " already declared!");
} else {
    if (initialNode == null) {
        type = constant(sizeNode);

        if (type.isIntLiteralType()) {
            AbstractRepresentation rep = null;

            symtab.setRepresentationRecord(identifNode.getText(), new RepresentationRecord());
            AbstractType repType = ((TypeRepresentation)
                symtab.getRepresentation(typeName)).getRepresentedType();

            rep = repType.instantiate();
            symtab.setRepresentation(identifNode.getText(), rep);
        }
        else {
            reportError("", ErrorCodeTable.INVALID_ARRAY_SIZE, ((SandboxAST) identifNode).getLine(), 0);
        }
    }
    else {
        reportError("", ErrorCodeTable.ARRAY_INITIALIZE, ((SandboxAST) identifNode).getLine(), 0);
    }
}
}

break;
}
default:
{
    throw new NoViableAltException(_t);
}
}
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}

```

```

    }
    _retTree = _t;
}

public final SandboxAST declareParameters(AST _t) throws RecognitionException {
    SandboxAST pList;

    AST declareParameters_AST_in = (AST)_t;
    AST paramList = null;

    pList = null;

    try {        // for error handling
        AST __t72 = _t;
        paramList = _t==ASTNULL ? null :(AST)_t;
        match(_t,PARAMETER_LIST);
        _t = _t.getFirstChild();
        {
        _loop74:
        do {
            if (_t==null) _t=ASTNULL;
            if ((_t.getType()==PARAMETER)) {
                declareParameter(_t);
                _t = _retTree;
            }
            else {
                break _loop74;
            }
        } while (true);
        }
        _t = __t72;
        _t = _t.getNextSibling();

        pList = (SandboxAST) paramList;
    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
    return pList;
}

public final void typeModifier(AST _t,
    java.lang.String varname
) throws RecognitionException {

    AST typeModifier_AST_in = (AST)_t;
    AST typeMod = null;

    try {        // for error handling
        typeMod = (AST)_t;
        match(_t,TYPE_MODIFIER);
        _t = _t.getNextSibling();

        SandboxAST modifierNode = (SandboxAST) typeMod.getFirstChild();
        if (modifierNode.getText().equals("const")) {
            RepresentationRecord r = syntab.getRepresentationRecord(varname);
            r.setConstant(true);
        } else {
            reportError("Don't know what to do with " + modifierNode.getText(), ErrorCodeTable.SEMANTIC_ERROR,
                modifierNode.getLine(), 0 );
        }
    }

    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

```

```

public final void declareParameter(AST _t) throws RecognitionException {

    AST declareParameter_AST_in = (AST)_t;
    AST formalNode = null;

    try {        // for error handling
        formalNode = (AST)_t;
        match(_t,PARAMETER);
        _t = _t.getNextSibling();

        AST typeNode = formalNode.getFirstChild();
        if (typeNode.getText().equals("TYPE_MODIFIER")) {
            typeNode = typeNode.getNextSibling();
        }

        AbstractRepresentation rep = symlab.getRepresentation(typeNode.getText());

        if (rep == null || !(rep.isTypeRepresentation())) {
            SandboxAST tNode = (SandboxAST) typeNode;
            reportError(typeNode.getText() + " is not a valid datatype.", ErrorCodeTable.NOT_TYPE, tNode.getLine(), 0);
            debugStream.println(typeNode.getText() + " is not a valid datatype.");
        }

    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final AbstractType primaryExpr(AST _t) throws RecognitionException {
    AbstractType type;

    AST primaryExpr_AST_in = (AST)_t;

    SandboxAST name;
    java.lang.String IDName;

    if (_t != null)
        debugStream.println(" Evaluating primaryExpr:" + _t.toStringList());
    else
        debugStream.println(" Evaluating primaryExpr: *** expression is null***");
    type = null;

    try {        // for error handling
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType()) {
        case IDENTIFIER:
            {
                IDName=identifier(_t);
                _t = _retTree;

                if (IDName != null) {
                    AbstractRepresentation rep = symlab.getRepresentation(IDName);
                    type = rep.getType();
                }

                break;
            }
        case FLOATING_CONSTANT:
        case FALSE:
        case TRUE:
        case INTEGER_CONSTANT:
        case CHAR_CONSTANT:
        case STRING_LITERAL:
            {
                type=constant(_t);
                _t = _retTree;
                break;
            }
        case LPAREN:
            {

```

```

        AST _t77 = _t;
        AST tmp13_AST_in = (AST)_t;
        match(_t,LPAREN);
        _t = _t.getFirstChild();
        type=expression(_t);
        _t = _retTree;
        _t = _t77;
        _t = _t.getNextSibling();
        break;
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
}
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return type;
}

public final java.lang.String identifier(AST _t) throws RecognitionException {
    java.lang.String type;

    AST identifier_AST_in = (AST)_t;
    AST idval = null;

    type = null;

    try { // for error handling
        idval = (AST)_t;
        match(_t,IDENTIFIER);
        _t = _t.getNextSibling();

        AbstractRepresentation rep = symtab.getRepresentation(idval.getText());

        debugStream.println(" Checking identifier: " + idval.getText());
        if (rep == null) {
            SandboxAST iNode = (SandboxAST) idval;

            reportError(idval.getText() + " has not been declared." ,
                ErrorCodeTable.NOT_DECLARED, iNode.getLine(), 0);
            debugStream.println(" " + idval.getText() + " not declared.");
        }
        // Check for identifier.field case. This check was inserted to keep Sandbox
        // from allowing code like int x; x.foo; to pass through. If we ever decide to
        // implement structs, this check will need to be removed and a better workaround
        // found.
        else if(idval.getNextSibling() != null && idval.getNextSibling().getText().equals(".")) {
            SandboxAST iNode = (SandboxAST) idval;
            reportError(idval.getText() + "." + idval.getNextSibling().getNextSibling().getText() +
                " has not been declared.", ErrorCodeTable.NOT_DECLARED, iNode.getLine(), 0);
            debugStream.println(" " + idval.getText() + "." +
                idval.getNextSibling().getNextSibling().getText() + " not declared.");
        }
        else {
            debugStream.println(" Identifier is of type " + rep.getType().getName());
            type = idval.getText();
        }
    }
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return type;
}

public final AbstractType constant(AST _t) throws RecognitionException {

```

```

AbstractType type;

AST constant_AST_in = (AST)_t;

    type = null;

try {      // for error handling
    if (_t==null) _t=ASTNULL;
    switch ( _t.getType() ) {
    case INTEGER_CONSTANT:
    {
        AST tmp14_AST_in = (AST)_t;
        match(_t,INTEGER_CONSTANT);
        _t = _t.getNextSibling();
        type = new IntLiteralType();
        break;
    }
    case FLOATING_CONSTANT:
    {
        AST tmp15_AST_in = (AST)_t;
        match(_t,FLOATING_CONSTANT);
        _t = _t.getNextSibling();
        type = new DoubleLiteralType();
        break;
    }
    case CHAR_CONSTANT:
    {
        AST tmp16_AST_in = (AST)_t;
        match(_t,CHAR_CONSTANT);
        _t = _t.getNextSibling();
        type = new CharLiteralType();
        break;
    }
    case STRING_LITERAL:
    {
        AST tmp17_AST_in = (AST)_t;
        match(_t,STRING_LITERAL);
        _t = _t.getNextSibling();
        type = new StringLiteralType();
        break;
    }
    case FALSE:
    case TRUE:
    {
        {
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType() ) {
            case TRUE:
            {
                AST tmp18_AST_in = (AST)_t;
                match(_t,TRUE);
                _t = _t.getNextSibling();
                break;
            }
            case FALSE:
            {
                AST tmp19_AST_in = (AST)_t;
                match(_t,FALSE);
                _t = _t.getNextSibling();
                break;
            }
            default:
            {
                throw new NoViableAltException(_t);
            }
            }
        }
        type = new BoolLiteralType();
        break;
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
}

```

```

    }
  }
}
catch (RecognitionException ex) {
  reportError(ex);
  if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return type;
}

public final void arguments(AST _t) throws RecognitionException {

  AST arguments_AST_in = (AST)_t;

  try {      // for error handling
    AST tmp20_AST_in = (AST)_t;
    match(_t,ARGUMENTS);
    _t = _t.getNextSibling();
  }
  catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
  }
  _retTree = _t;
}

public static final String[] _tokenNames = {
  "<0>",
  "EOF",
  "<2>",
  "NULL_TREE_LOOKAHEAD",
  "DOT",
  "FLOATING_CONSTANT",
  "break",
  "case",
  "const",
  "const_cast",
  "continue",
  "default",
  "do",
  "else",
  "enum",
  "for",
  "false",
  "if",
  "namespace",
  "using",
  "return",
  "reinterpret_cast",
  "sizeof",
  "static_cast",
  "struct",
  "switch",
  "TYPENAME",
  "typedef",
  "true",
  "while",
  "LBRACKET",
  "RBRACKET",
  "LPAREN",
  "RPAREN",
  "LBRACE",
  "RBRACE",
  "INCR",
  "DECR",
  "AMPERSAND",
  "STAR",
  "PLUS",
  "MINUS",
  "TILDE",
  "BANG",
  "SLASH",

```

```
"PERCENT",
"LSHIFT",
"RSHIFT",
"LT",
"GT",
"LE",
"GE",
"EQ",
"NE",
"CARET",
"PIPE",
"LAND",
"LOR",
"QUESTION",
"COLON",
"COLON_COLON",
"SEMI",
"ASSIGN",
"STAR_ASSIGN",
"SLASH_ASSIGN",
"PERCENT_ASSIGN",
"PLUS_ASSIGN",
"MINUS_ASSIGN",
"COMMA",
"POUND",
"LSHIFT_ASSIGN",
"RSHIFT_ASSIGN",
"AMPERSAND_ASSIGN",
"CARET_ASSIGN",
"PIPE_ASSIGN",
"WS",
"WHITESPACE",
"NEW_LINE",
"SL_COMMENT",
"ML_COMMENT",
"IDENTIFIER",
"IDENTIFIER_NONDIGIT",
"DIGIT",
"INTEGER_CONSTANT",
"EXPONENT",
"FLOATING_SUFFIX",
"INTEGER_SUFFIX",
"CHAR_CONSTANT",
"STRING_LITERAL",
"ESC",
"HEADER_NAME",
"PP_TOKENS",
"PREPROCESSING_TOKEN",
"POUND_INCLUDE_DIRECTIVE",
"ARGUMENTS",
"ARRAY_DECLARATOR",
"BLOCK_STATEMENT",
"CALL",
"CASE_GROUP",
"CAST",
"DECLARATION",
"DECLARATOR",
"ENUMERATOR_LIST",
"EXPRESSION_STATEMENT",
"FOR_INIT",
"FOR_COND",
"FOR_ITER",
"FUNCTION",
"FUNCTION_PROTOTYPE",
"INCLUDE",
"INITIALIZER",
"PARAMETER",
"PARAMETER_LIST",
"POST_DECR",
"POST_INCR",
"PROGRAM",
"SLIST",
"STRUCT_DECLARATION",
"STRUCT_MEMBERS",
```

```
"SUBSCRIPT",  
"TYPE_MODIFIER",  
"TYPE_SIZEOF",  
"USING_DIRECTIVE"  
};  
  
private static final long _tokenSet_0_data_[] = { 571642944L, 622770257920L, 0L, 0L };  
public static final BitSet _tokenSet_0 = new BitSet(_tokenSet_0_data_);  
}
```

J.7 ANTLR Input File Used to Create *CS1 Sandbox's* Interpreter (Interp.g)

```

header {
package sandbox.compiler.backend;

import java.io.*;
import java.util.*;
import java.lang.reflect.*;
import sandbox.client.*;
import sandbox.compiler.*;
import sandbox.compiler.type.*;
import sandbox.compiler.syntab.*;
import sandbox.compiler.frontend.*;
import sandbox.lib.*;
import sandbox.lib.string.*;
import sandbox.lib.iostream.*;
}

/**
 * NewInterp is the last step in the interpretation process.
 * After having gone through the parser, and passed the semantic check,
 * we are now ready to execute the program.
 */

class NewInterp extends TreeParser;
options {

importVocab = CLexer;
importVocab = CParser;
}

{
// Items related to the debugging stream
//
// If you want to put an error message to the debug stream
// do something like this: debugStream.println("This is my error.");
ByteArrayOutputStream out = new ByteArrayOutputStream();
PrintStream debugStream = new PrintStream(out);

// Create the symbol table as a "global" variable.
SymTab syntab = new LinkSymTab();

private LibraryManager libraryManager = null;
private TypeManager typeManager = null;
private AbstractRepresentation ONE = new IntRepresentation(1);
private MenuButtonListener threadKiller = null;
private IOFrame iof = null;

// The function manager, manages the addition, use, and query of user-defined functions
private FunctionManager fm;

// Holds the wrapper to the function under execution.
// private FunctionWrapper currentFunction = null;
private Stack functionStack = new Stack();

// Used to track current line numbers for run-time errors (when direct access to the AST node
// is not possible and passing the node means changing a lot of code)
private int currentLineNumber = 0;

/**
 * Constructor which accepts a reference to the function manager.
 */
public NewInterp (FunctionManager funcMgr) {
fm = funcMgr;
}

/**

```

```
* Sets the library manager to the parameter, also sets the
* reference to the type manager.
*/
public void setLibraryManager(LibraryManager mgr) {
    libraryManager = mgr;
    typeManager = libraryManager.getTypeManager();
}

/**
 * Registers the IOFrame used for standard input (cin) and standard output (cout)
 */
public void setIOFrame(IOFrame iof) {
    this.iof = iof;
}

/** The errors that are observed during the parse phase. */
private Vector errorVector = new Vector();

/**
 * Returns a vector of each error seen during compilation.
 */
public Vector getErrorVector() {
    return(errorVector);
}

public void reportError (int code, int line, int col) {
    reportError("", code, line, col);
}

public void reportError (java.lang.String errMsg, int code, int line, int col) {
    AntlrError e = new AntlrError ();
    e.setType("Interpreter error");
    e.setErrorMessage(errMsg);
    e.setErrorCode(code);
    e.setLineNumber(line);
    e.setColumnNumber(col);
    errorVector.add(e);
}

public void reportError(RecognitionException ex) {
    AntlrError error = new AntlrError();
    error.setType("recognition exception -- interpreter");
    error.setErrorMessage(ex.getMessage());
    error.setLineNumber(ex.getLine());
    error.setColumnNumber(ex.getColumn());
    errorVector.add(error);
}

public void reportError(java.lang.String s) {
    AntlrError error = new AntlrError();
    error.setType("error");
    error.setErrorMessage(s);
    errorVector.add(error);
}

public void reportWarning(java.lang.String s) {
    AntlrError error = new AntlrError();
    error.setType("warning");
    error.setErrorMessage(s);
    errorVector.add(error);
}

/**
 * Sets this interpreter's debugging PrintStream object
 */
```

```

public void setDebugOutput(PrintStream p) {
    debugStream = p;
}

/**
 * Returns the OutputStream for this interpreter
 */
public java.lang.String getDebugOutput() {
    return(out.toString());
}

/**
 * Returns the string name of the type for identifier 'identifName', assuming it exists.
 */
private java.lang.String getType(java.lang.String identifName) {
    java.lang.String returnTypeName = null;
    if (symtab == null) System.err.println("Symtab is null in interp.");

    if ((symtab != null) && (identifName != null)) {
        AbstractRepresentation abRep = symtab.getRepresentation(identifName);
        if (abRep != null)
            returnTypeName = abRep.getType().getName();
    }
    return returnTypeName;
}

/**
 * Builds a new type map for the primitive types and ensures they are placed in the
 * symbol table.
 */
protected void assessTypes() {
    typeManager.buildTypeMap();
    for(Enumeration e = typeManager.getTypeNames(); e.hasMoreElements(); ) {
        java.lang.String typeName = (java.lang.String) e.nextElement();
        AbstractType type = typeManager.getType(typeName);
        TypeRepresentation rep = new TypeRepresentation(type);

        symtab.setRepresentationRecord(typeName, new RepresentationRecord());
        symtab.setRepresentation(typeName, rep);
    }
}

/**
 * Evaluates a boolean expression (!, &&, ||) and returns the appropriate result.
 */
protected boolean evaluateBoolExpression(AST node) throws RecognitionException {
    AbstractRepresentation rep = expression(node);
    try {
        // Try to convert to a boolean
        BoolRepresentation bRep = (BoolRepresentation) typeManager.convert(rep, new BoolType());

        return bRep.getValue();
    } catch(Exception ex) {
        if (rep.getType().getName().equals("istream") || rep.getType().getName().equals("ifstream")) {
            return !((IStreamRepresentation) rep).getFailed();
        } else if (rep.getType().getName().equals("ostream") || rep.getType().getName().equals("ofstream"))
            return !((OStreamRepresentation) rep).getFailed();
        else
            return false;
    }
}

/**
 * Evaluates a given expression (generally a test expression used in for loops, while loops,
 * * and if statements) and returns a boolean true or false value.
 */
protected boolean evaluateTestExpression(AST node) throws RecognitionException {
    AbstractRepresentation rep = expression(node);
    try {

```

```

    // Is the representation a stream??
    AbstractType type = rep.getType();
    if (type.isStream()) {
        // try to convert to a stream object and return if the stream has failed.
        Stream stream = (Stream) rep;
        return (!stream.getFailed());
    }
    else { // the value should be numeric
        // Try to convert to an integer
        IntRepresentation iRep = (IntRepresentation)typeManager.convert(rep, new IntType());
        return (iRep.getValue() != 0);
    }
} catch(Exception ex) {
    ex.printStackTrace();
}

return false;
}

/**
 * Attempts to evaluate a binary operation expression given the operator ('+', '-', '*', etc.)
 * and the left-hand side (lhs) and the right-hand side (rhs). Really passes this to the type
 * manager for execution (interpretation).
 */
protected AbstractRepresentation evaluateBinaryOp(int operator, AbstractRepresentation lhs, AbstractRepresentation rhs) {
    try {

        debugStream.print("  binary op " + operator + " on " + lhs.getType().getName() + " and " + rhs.getType().getName());
        AbstractRepresentation rep = typeManager.evaluateBinaryOp(operator, lhs, rhs);
        if (rep != null)
            debugStream.println("...result is " + rep.getType().getName());

        return rep;
    } catch(Exception ex) {
        // The following section catches a negative assignment and returns the correct value
        // and also catches the divide by zero problem and sets a run-time error message.
        if (operator == MINUS) {
            try {
                rhs.getType();
            } catch(Exception ex2) {
                return lhs.negate(); // negates the current value - Dave 2/14/02
            }
        } else if ((operator == SLASH || operator == PERCENT) && ex.getMessage().equals("/ by zero")) {
            reportError(null, ErrorCodeTable.DIVIDE_BY_ZERO, currentLineNumber, 0);
            debugStream.println("  Run time error: divide by zero.");
            return null;
        }
        debugStream.println("...exception caught");
        ex.printStackTrace();
        return null;
    }
}

/**
 * Attempts to evaluate an assignment expression given the operator (remember that assignments
 * are not just '=', but '+=', '-=', etc.) and the left-hand side (lhs) and the right-hand
 * side (rhs). An implicit type conversion (type coercion) takes place first (if needed).
 */
protected AbstractRepresentation evaluateAssignment(int op, AbstractRepresentation lhs, AbstractRepresentation rhs) {
    try {

        rhs = typeManager.convert(rhs, lhs.getType());
        lhs.setValue(rhs);
    } catch(Exception ex) {
        ex.printStackTrace();
    }
    finally {
        return rhs;
    }
}

```

```

/**
 * Attempts to evaluate the passed in value by determining if it is a constant
 * (character constant [e.g., '.'], string literal [e.g., "hi"], etc..). If the
 * variable is not a constant then it must be an identifier. The value of the
 * identifier is then determined and returned.
 */
protected AbstractRepresentation evaluateVariable(SandboxAST value, AbstractType type) throws IncompatibleTypesException {
    AbstractRepresentation rep= null; // The representation of the value
    try {
        // test if the value is a literal
        rep = constant(value);
    } catch (RecognitionException re) {
        System.err.println("Recognition exception in evaluateVariable");
        re.printStackTrace();
    }

    if (rep == null) { // it is not a constant, it must be an identifier
        rep = symtab.getRepresentation(value.getText());
    }

    return typeManager.convert(rep, type);
}

/**
 * Sets a reference to the MenuButtonListener so we can check safely for an interruption
 * of this thread (in the event of endless loops!)
 */
public void setListener(MenuButtonListener mbl) {
    threadKiller = mbl;
}

/**
 * Returns an array of class objects of actual parameter types to a library function call.
 * ctuals is assumed to be the ARGUMENTS node in the AST, so we first need to skip down
 * to the first parameter.
 */
protected Class[] getActualTypes(SandboxAST actuals) {
    if(actuals == null)
        return new Class[] {};

    SandboxAST param = (SandboxAST) actuals.getFirstChild();
    ArrayList tempList = new ArrayList();

    while (param != null) {
        try {
            AbstractRepresentation rep = expression((AST) param);
            if (rep instanceof DoubleRepresentation || rep instanceof FloatRepresentation)
                tempList.add(DoubleRepresentation.class);

            else if (rep instanceof IntRepresentation || rep instanceof LongRepresentation ||
                    rep instanceof ShortRepresentation)
                tempList.add(IntRepresentation.class);

            else if (rep instanceof BoolRepresentation)
                tempList.add(BoolRepresentation.class);

            else if (rep instanceof CharRepresentation)
                tempList.add(CharRepresentation.class);

            else if (rep instanceof IStreamRepresentation || rep instanceof IFStreamRepresentation)
                tempList.add(IStreamRepresentation.class);

            else if (rep instanceof StringRepresentation)
                tempList.add(sandbox.lib.string.StringRepresentation.class);

            else if (rep instanceof StringLiteralRepresentation)
                tempList.add(StringLiteralRepresentation.class);

            param = (SandboxAST) param.getNextSibling();
        } catch (RecognitionException re) {
            re.printStackTrace();
        }
    }
}

```

```

    }

    // Build and return the list of class objects
    int count = tempList.size();
    Class[] classArray = new Class[count];
    for (int i=0; i < count; i++)
        classArray[i] = (Class) tempList.get(i);

    return classArray;
}

/**
 * Returns an array of objects of the values of the actual parameters to a library function call.
 * Primitive data type parameters must be wrapped in wrapper classes to utilize Java
 * reflection, needed to call the library function. Actuals is assumed to be the ARGUMENTS
 * node in the AST, so we first need to skip down to the first parameter.
 */
protected Object[] getActualValues(SandboxAST actuals) {
    if(actuals == null)
        return new Object[] {};

    SandboxAST param = (SandboxAST) actuals.getFirstChild();
    ArrayList tempList = new ArrayList();

    while (param != null) {
        try {
            AbstractRepresentation rep = expression((AST) param);

/*            if (rep instanceof DoubleRepresentation)
                tempList.add(new Double(((DoubleRepresentation) rep).getValue()));

            else if (rep instanceof FloatRepresentation)
                tempList.add(new Double(((FloatRepresentation) rep).getValue()));

            else if (rep instanceof LongRepresentation)
                tempList.add(new Integer((int)((LongRepresentation) rep).getValue()));

            else if (rep instanceof IntRepresentation)
                tempList.add(new Integer(((IntRepresentation) rep).getValue()));

            else if (rep instanceof BoolRepresentation)
                tempList.add(new Boolean(((BoolRepresentation) rep).getValue()));

            else if (rep instanceof CharRepresentation)
                tempList.add((CharRepresentation) rep);

            else if ((rep instanceof IStreamRepresentation) || (rep instanceof IFStreamRepresentation))
                tempList.add((IStreamRepresentation) rep);

            else if (rep instanceof StringLiteralRepresentation)
                tempList.add((StringLiteralRepresentation) rep);

*/

            if (rep instanceof StringRepresentation)
                if (((StringRepresentation) rep).getValue() == null)
                    tempList.add(new StringRepresentation(""));
                else
                    tempList.add((StringRepresentation) rep);
            else if (rep instanceof LongRepresentation)
                tempList.add(new IntRepresentation(((LongRepresentation) rep).getValue()));
            else if (rep instanceof ShortRepresentation)
                tempList.add(new IntRepresentation(((ShortRepresentation) rep).getValue()));
            else
                tempList.add( rep );

            param = (SandboxAST) param.getNextSibling();

        } catch (RecognitionException re) {
            re.printStackTrace();
        }
    }

    // Build and return the list of objects (wrappers)

```

```

int count = tempList.size();
Object[] actualsArray = new Object[count];
for (int i=0; i < count; i++)
    actualsArray[i] = tempList.get(i);

return actualsArray;
}

/**
 * Returns an array of objects of the values of the actual parameters to a library function call.
 * Primitive data type parameters must be wrapped in wrapper classes to utilize Java
 * reflection, needed to call the library function. Actuals is assumed to be the ARGUMENTS
 * node in the AST, so we first need to skip down to the first parameter.
 * This method differs from the one above as it performs the coercion as it proceeds.
 */
protected Object[] getConvertedActualValues(SandboxAST actuals, Class[] formals) {
    SandboxAST param = (SandboxAST) actuals.getFirstChild();
    ArrayList tempList = new ArrayList();
    int index = 0;

    while (param != null) {
        try {
            AbstractRepresentation rep = expression((AST) param);

            if (formals[index].getName().equals("java.lang.Double")) {
                DoubleRepresentation dRep = (DoubleRepresentation) typeManager.convert(rep, new DoubleType());
                tempList.add(new Double(dRep.getValue()));
            }
            else if (formals[index].getName().equals("java.lang.Integer")) {
                IntRepresentation iRep = (IntRepresentation) typeManager.convert(rep, new IntType());
                tempList.add(new Integer(iRep.getValue()));
            }
            else if (formals[index].getName().equals("java.lang.Boolean")) {
                BoolRepresentation bRep = (BoolRepresentation) typeManager.convert(rep, new BoolType());
                tempList.add(new Boolean(bRep.getValue()));
            }
            else if (formals[index].getName().equals("java.lang.Character")) {
                CharRepresentation cRep = (CharRepresentation) typeManager.convert(rep, new CharType());
                tempList.add(new Character((char) cRep.getValue()));
            }
            else if (formals[index].getName().equals("sandbox.compiler.type.DoubleRepresentation")) {
                DoubleRepresentation dRep = (DoubleRepresentation) typeManager.convert(rep, new DoubleType());
                tempList.add(dRep);
            }
            else if (formals[index].getName().equals("sandbox.compiler.type.IntRepresentation")) {
                IntRepresentation iRep = (IntRepresentation) typeManager.convert(rep, new IntType());
                tempList.add(iRep);
            }
            param = (SandboxAST) param.getNextSibling();
            index++;
        } catch (RecognitionException re) {
            re.printStackTrace();
        } catch (IncompatibleTypesException ite) {
            // this should not happen as well, if we have gotten to the interp...
            ite.printStackTrace();
        }
    }

    // Build and return the list of objects (wrappers)
    int count = tempList.size();
    Object[] actualsArray = new Object[count];
    for (int i=0; i < count; i++)
        actualsArray[i] = tempList.get(i);

    return actualsArray;
}

/**
 * Converts the src type (a Java class, as in Integer.class) to the corresponding
 * internal primitive type (IntType) for use in calculating type conversions when
 * looking up library function calls.
 */

```

```

public static AbstractRepresentation convertClassToAbstract(Class src) {
    if (src.getName().equals("java.lang.Integer"))
        return new IntRepresentation();
    else if (src.getName().equals("java.lang.Double"))
        return new DoubleRepresentation();
    else if (src.getName().equals("java.lang.Character"))
        return new CharRepresentation();
    else if (src.getName().equals("java.lang.Boolean"))
        return new BoolRepresentation();
    else
        return null; // this should never occur!
}
}

program
{
    assessTypes();
}
: #(PROGRAM (include)* (using)* (declaration | functDef | functionPrototype)*
{
    FunctionWrapper function = fm.locateFunction("main", 0);
    AbstractRepresentation[] params = { new IntRepresentation(0) };

    symtab.setRepresentationRecord("__return", new RepresentationRecord());
    function.invoke(this, params);
}
;

using
: USING_DIRECTIVE
// Politely ignore the using directive
;

include
: includeNode:INCLUDE
{
    try {
        java.lang.String libName = includeNode.getFirstChild().getText().trim();
        Library lib = libraryManager.loadLibrary(libName);

        assessTypes();
        if (libName.equals("iostream"))
            lib.loadSymbols(symtab, iof);
        else
            lib.loadSymbols(symtab);
    } catch(Exception ex) {
        ex.printStackTrace();
    }
}
;

functDef
: FUNCTION
{
    /**
     * There is nothing here since all functions are added to the
     * Function Manager now.... this is not needed. Functions are placed in the
     * manager in the semantic pass, so there's nothing here to do!
     */
}
;

functionPrototype
: FUNCTION_PROTOTYPE
{
    /**
     * There is nothing here since all functions are added to the
     * Function Manager now.... this is not needed. Function prototypes are placed in the
     * manager in the semantic pass, so there's nothing here to do!
     */
}

```

```

;

function [AbstractRepresentation[] params] returns [AbstractRepresentation retval]
{
    // The parameter 'params' is the list of the actual parameters

    retval = null;
}
: funcNode:FUNCTION
{
    symtab.beginFunction();

    AST type = funcNode.getFirstChild(); // references the node containing the return type
    AST name = type.getNextSibling(); // references the node containing the name of the function
    AST tempNode = name.getNextSibling(); // references the next node, this is either the parameters or the body (block)
    AST paramsNode = null; // references the start of the formal parameters list
    AST block = null;
    int fParamCount = 0;

    debugStream.println("Interpreting function: " + name.getText());

    if (!tempNode.getText().equals("BLOCK_STATEMENT")) {
        paramsNode = name.getNextSibling();
        fParamCount = formalParameterList(paramsNode, params);
        block = paramsNode.getNextSibling();
    } else
        block = tempNode;

    // Set the currentFunction to the function wrapper
    // currentFunction = fm.locateFunction(name.getText(), fParamCount);
    functionStack.push(fm.locateFunction(name.getText(), fParamCount));

    java.lang.String exitCode = statement(block);
    if (exitCode != null && exitCode.equals("return"))
        retval = symtab.getRepresentation("__return");

    symtab.endFunction();
}
;

formalParameterList [AbstractRepresentation[] actuals] returns [int params]
{
    // The parameter 'actuals' is the list of the actual parameters

    params = 0;
}
: paramsNode:PARAMETER_LIST
{
    AST singleParam = paramsNode.getFirstChild();

    while (singleParam != null) {
        formalParameter(singleParam, actuals[params]);
        singleParam = singleParam.getNextSibling();
        params++;
    }
}
;

formalParameter [AbstractRepresentation actual]
{
    // The parameter 'actual' is the representation of an actual parameter

    AST refNode = null;
    AbstractRepresentation rep;
}
: formalNode:PARAMETER
{
    AST typeNode = formalNode.getFirstChild();
    if (typeNode.getText().equals("TYPE_MODIFIER"))
        typeNode = typeNode.getNextSibling();

    AST paramNode = typeNode.getNextSibling();
    if (paramNode != null && paramNode.getText().equals("&")) {

```

```

        refNode = paramNode;
        paramNode = paramNode.getNextSibling();
        rep = actual;
    } else
        rep = actual.cloneRepresentation();

    // Create the representation record for this entry in the symbol table.
    RepresentationRecord newRR = new RepresentationRecord();

    // If the actual is really a constant, set the formal to be a constant
    // unbeknownst to the user!
    if (actual.isConstant())
        newRR.setConstant(true);

    // Add this parameter to the symbol table.
    symtab.setRepresentationRecord(paramNode.getText(), newRR);
    symtab.setRepresentation(paramNode.getText(), rep);
    debugStream.println(" Formal parameter " + paramNode.getText() + " set to " + rep);
}
;

/*-----
STATEMENTS PART
-----*/
statement returns [java.lang.String exitCode]
{
    if (threadKiller.getStopThread())
        return "killThread";
    exitCode = null;
}
: exitCode=whileStatement
| exitCode=forStatement
| exitCode=doStatement
| exitCode=ifStatement
| exitCode=switchStatement
| expressionStatement
| exitCode=blockStatement
| declaration
| exitCode=returnStatement
| exitCode=breakStatement
| exitCode=continueStatement
;

switchStatement returns [java.lang.String exitCode]
{
    exitCode = null;
}
: switchNode:SWITCH
{
    AST exprNode = switchNode.getFirstChild();
    AbstractRepresentation as = symtab.getRepresentation(exprNode.toString());
    AbstractRepresentation abRep = null;
    java.lang.String typeName;

    if (as != null)
        typeName = as.getType().getName();
    else
        typeName = "expression";

    java.lang.String valueToMatch;
    if (typeName.equals("char"))
        valueToMatch = "'" + ((CharRepresentation) as).toString() + "'";
    else if (typeName.equals("int"))
        valueToMatch = new java.lang.String((new Integer(((IntRepresentation) as).getValue())).toString());
    else if (typeName.equals("expression")) {
        abRep = expression(exprNode);
        long vTM = ((LongRepresentation) abRep).getValue();
        valueToMatch = new java.lang.String((new Long(vTM)).toString());
    } else
        return exitCode;

    debugStream.println(" Attempting to match: " + valueToMatch);
}

```

```

syntab.beginLevel();

boolean breakFound = false;
boolean flowThrough = false;
AST groupNode = exprNode.getNextSibling(); // these should be CASE_GROUP nodes
while(groupNode != null && !breakFound) { // iterate over the CASE_GROUP nodes
    AST labelNode = groupNode.getFirstChild();
    while (labelNode != null && !breakFound) {
        if (labelNode.getText().equals("case")) {
            AST labelValue = labelNode.getFirstChild();

            debugStream.println("    Examining case statement: " + labelValue.getText());
            debugStream.println("    Attempting to match: " + valueToMatch + " and " + labelValue.getText());

            // Clean up comparison here, take into account char and int literals
            // (not a problem, as much since we use the string value of the node
            // and obtain the value of a char or int constant identifier.... something we never did....
            String labelType = getType(labelValue.getText());
            String labelToMatch = null;

            // Obtain the label's value, be wary of literals and constant identifiers!
            if (labelType != null) {
                if (labelType.equals("int")) {
                    // Handle case of constant int identifier
                    IntRepresentation intRep = ((IntRepresentation) syntab.getRepresentation(labelValue.getText()));
                    labelToMatch = (new Integer(intRep.getValue())).toString();

                } else if (labelType.equals("char")) {
                    // Handle case of constant character identifier
                    CharRepresentation charRep = ((CharRepresentation) syntab.getRepresentation(labelValue.getText()));
                    labelToMatch = '\'' + (new Character((char) charRep.getValue())).toString() + '\'';
                }
            } else if (labelType == null) {
                // It is a char or int literal, so use the node's value

                // If it's a char literal, the second char is a ',
                if (labelValue.getText().charAt(0) == '\''
                    && labelValue.getText().charAt(1) == '\\')
                    switch (labelValue.getText().charAt(2)) {
                        case 'n':
                            labelToMatch = "\n";
                            break;

                        case 'r':
                            labelToMatch = "\r";
                            break;

                        case 't':
                            labelToMatch = "\t";
                            break;
                    }
                else
                    labelToMatch = labelValue.getText();
            } else
                labelToMatch = labelValue.getText();
        }
    }
    if (valueToMatch.equals(labelToMatch) || flowThrough) {
        // Match found, skip until we see the SList and then process the list.
        debugStream.println("    Executing case statement: " + labelValue.toString());
        labelNode = labelNode.getNextSibling();
        while (labelNode != null && !labelNode.getText().equals("SLIST"))
            labelNode = labelNode.getNextSibling();

        // this should never be null, there should always be a SLIST to find
        if (labelNode != null && labelNode.getText().equals("SLIST")) {
            AST SListNode = labelNode.getFirstChild();
            while (SListNode != null && !breakFound) {
                exitCode = statement(SListNode);
                if (exitCode != null && (exitCode.equals("break") || exitCode.equals("return"))) {
                    breakFound = true;
                    flowThrough = false;
                } else
            }
        }
    }
}

```

```

        flowThrough = true;
        SListNode = SListNode.getNextSibling();
    }
}
} else if (labelNode.getText().equals("default")) {
    debugStream.println("  Executing default statement.");
    AST SListNode = labelNode.getNextSibling().getFirstChild(); // picks up the first statement of the SLIST
    while (SListNode != null) {
        exitCode = statement(SListNode);
        SListNode = SListNode.getNextSibling();
    }
    labelNode = labelNode.getNextSibling();
}
groupNode = groupNode.getNextSibling();
}
syntab.endLevel();

// Reset the exit code unless we saw a return in there... (from actual student implementation)
if (exitCode != null && !exitCode.equals("return"))
    exitCode = null;
}
;

returnStatement returns [java.lang.String exitCode]
{
    exitCode = "return";
    AbstractRepresentation expr = null;
    AbstractRepresentation rep = null;
}
: #(RETURN (expr=expression)?)
{
    FunctionWrapper currentFunction = (FunctionWrapper) functionStack.pop();
    AbstractType typeToReturn = ((TypeRepresentation) syntab.getRepresentation(
        currentFunction.getReturnType())).getRepresentedType();
    try {
        if (expr != null)
            rep = typeManager.convert(expr, typeToReturn);
    } catch (IncompatibleTypesException ex) {
    }
    syntab.setRepresentation("__return", rep);
}
;

breakStatement returns [java.lang.String exitCode]
{
    exitCode = "break";
}
: BREAK
;

continueStatement returns [java.lang.String exitCode]
{
    exitCode = "continue";
}
: CONTINUE
;

blockStatement returns [java.lang.String exitCode]
{
    exitCode = null;
}
: blockNode:BLOCK_STATEMENT
{
    AST stmtNode;
    stmtNode = blockNode.getFirstChild();
    syntab.beginLevel();
    while (stmtNode != null) {
        debugStream.println("Executing statement: " + stmtNode);
    }
}
;

```



```

    } else if (forIter.getFirstChild() == null) {
        // Init not null, Iter null
        for(expression(forInit.getFirstChild()); evaluateTestExpression(forCond.getFirstChild()); ) {
            if (bodyNode != null) {
                exitCode = statement(bodyNode);
                if (exitCode != null) {
                    if (exitCode.equals("return"))
                        break;
                    else if (exitCode.equals("break")) {
                        exitCode = null;
                        break;
                    } else if (exitCode.equals("continue"))
                        exitCode = null;
                }
            }
        }
    }
} else if (forInit.getFirstChild() == null) {
    if (forIter.getFirstChild() != null) {
        // Init null, Iter not null
        for( ; evaluateTestExpression(forCond.getFirstChild()); expression(forIter.getFirstChild())) {
            if (bodyNode != null) {
                exitCode = statement(bodyNode);
                if (exitCode != null) {
                    if (exitCode.equals("return"))
                        break;
                    else if (exitCode.equals("break")) {
                        exitCode = null;
                        break;
                    } else if (exitCode.equals("continue"))
                        exitCode = null;
                }
            }
        }
    }
} else if (forIter.getFirstChild() == null) {
    // Init null, Iter null
    for( ; evaluateTestExpression(forCond.getFirstChild()); ) {
        if (bodyNode != null) {
            exitCode = statement(bodyNode);
            if (exitCode != null) {
                if (exitCode.equals("return"))
                    break;
                else if (exitCode.equals("break")) {
                    exitCode = null;
                    break;
                } else if (exitCode.equals("continue"))
                    exitCode = null;
            }
        }
    }
}
}
}
}
}
;

ifStatement returns [java.lang.String exitCode]
{
    exitCode = null;
}
: ifNode:IF
{
    debugStream.println("Executing if statement");
    AST exprNode = ifNode.getFirstChild();
    AST trueNode = exprNode.getNextSibling();
    AST falseNode = trueNode.getNextSibling();
    boolean result = evaluateTestExpression(exprNode);

    if (result) {
        debugStream.println(" Following the true branch:");
        exitCode = statement(trueNode);
    } else if (falseNode != null) {
        debugStream.println(" Following the false branch:");
    }
}

```

```

        exitCode=statement(falseNode);
    } else {
        debugStream.println("    Following the false, but no branch exists.");
        exitCode = null;
    }
}
;

whileStatement returns [java.lang.String exitCode]
{
    exitCode = null;
}
: whileNode:WHILE
    {
        debugStream.println("Executing while statement");
        AST exprNode = whileNode.getFirstChild();
        AST stmtNode = exprNode.getNextSibling();

        while (evaluateTestExpression(exprNode)) {
            if (stmtNode != null) {
                exitCode = statement(stmtNode);
                if (exitCode != null) {
                    if (exitCode.equals("return"))
                        break;
                    else if (exitCode.equals("break")) {
                        exitCode = null;
                        break;
                    } else if (exitCode.equals("continue")) {
                        exitCode = null;
                        continue;
                    }
                }
            }
        }
    }
;

doStatement returns [java.lang.String exitCode]
{
    exitCode = null;
}
: doNode:DO
    {
        debugStream.println("Executing while statement");
        AST bodyNode = doNode.getFirstChild();
        AST exprNode = bodyNode.getNextSibling();

        do {
            if (bodyNode != null) {
                exitCode = statement(bodyNode);
                if (exitCode != null) {
                    if (exitCode.equals("return")) {
                        break;
                    } else if (exitCode.equals("break")) {
                        exitCode = null;
                        break;
                    } else if (exitCode.equals("continue")) {
                        exitCode = null;
                        continue;
                    }
                }
            }
        }
        //Evaluate do condition...
    } while(evaluateTestExpression(exprNode));
;

expressionStatement
{
    // this item is only used to store the value so that no warning is thrown during compile time
    AbstractRepresentation z = null;
}

```

```

}
: #(EXPRESSION_STATEMENT z=expression)
;

/*-----
DECLARATION PART
-----*/
declaration
: declNode:DECLARATION
{
  AST modifierNode = declNode.getFirstChild();
  AST typeNode;
  boolean isConst = false;

  if (modifierNode.getText().equals("TYPE_MODIFIER")) {
    if (modifierNode.getFirstChild().getText().equals("const"))
      isConst = true;

    typeNode = modifierNode.getNextSibling();
  } else {
    typeNode = modifierNode;
    modifierNode = null;
  }
  AST declaratorNode = typeNode.getNextSibling();

  while (declaratorNode != null) {
    declarator(declaratorNode, typeNode.getText(), isConst);
    declaratorNode = declaratorNode.getNextSibling();
  }
}
;

declarator [java.lang.String typeName, boolean isAConst]
{
  AbstractRepresentation s = null;
}
: #(DECLARATOR id:IDENTIFIER (s=expression)?)
{
  AbstractRepresentation typeRep = symtab.getRepresentation(typeName);
  AbstractRepresentation rep = null;
  RepresentationRecord rr = new RepresentationRecord();
  if (isAConst)
    rr.setConstant(true);

  symtab.setRepresentationRecord(id.getText(), rr);

  if (typeRep == null || !(typeRep instanceof TypeRepresentation))
    debugStream.println(" Unknown type: " + typeName);
  else {
    AbstractType type = ((TypeRepresentation) typeRep).getRepresentedType();

    if (s == null)
      rep = type.instantiate();
    else
      try {
        debugStream.println(" Initializing " + id.getText() + " to: " + s.toString());
        rep = typeManager.convert(s, type);
      } catch(IncompatibleTypesException ex) {
        debugStream.println(" Could not cast initializer!");
      }
  }
  symtab.setRepresentation(id.toString(), rep);
  debugStream.println(" Declared " + id.getText() + " as " + rep);
}
;

/*-----
EXPRESSION PART
-----*/
primaryExpr returns [AbstractRepresentation retval]
{

```

```

    debugStream.println("    evaluating primaryExpr: " + _t.toStringList());
    retval = null;
}
: idval:IDENTIFIER
  {
    retval = sytab.getRepresentation(idval.getText());
  }
| retval=constant
| #(LPAREN retval=expression)
;

constant returns [AbstractRepresentation retval]
{
  retval = null;
}
: intval:INTEGER_CONSTANT
  {
    retval = new LongRepresentation(intval.getText());
  }
| floatval:FLOATING_CONSTANT
  {
    retval = new DoubleRepresentation(floatval.getText());
  }
| charval:CHAR_CONSTANT
  {
    retval = new CharRepresentation(EscapeTranslator.translateEscapeSequence(charval.getText()).charAt(1));
  }
| stringval:STRING_LITERAL
  {
    java.lang.String str = EscapeTranslator.translateEscapeSequence(stringval.getText());
    retval = new StringLiteralRepresentation(str.substring(1, str.length() - 1));
  }
| TRUE
  {
    retval = new BoolRepresentation(true);
  }
| FALSE
  {
    retval = new BoolRepresentation(false);
  }
;

/*----
arguments perhaps needs some explanation, since we almost deleted the
darn thing! Arguments are the "actual parameters part" this parses the
parameter arguments of a function CALL, not function definition, and so
is not redundant.
----*/
arguments returns [AbstractRepresentation[] repArray]
{
  repArray = null;
}
: argNode:ARGUMENTS
  {
    Vector repVector = new Vector();
    AST singleArg = argNode.getFirstChild();
    AbstractRepresentation rep;

    while(singleArg != null) {
      rep = expression(singleArg);

      // Set the AbstractRepresentation to be a constant if the argument is a constant.
      // This is a semi-duplication of the constant flag, but since we can't get access
      // to the argument's name at the time of the call, we need to pass the info through
      // the AbstractRepresentation. Used to check functions attempting to modify a constant
      // when passed by reference.
      RepresentationRecord rr = sytab.getRepresentationRecord(singleArg.getText());
      if (rr != null && rr.isConstant())
        rep.setConstant();

      repVector.addElement(rep);
      singleArg = singleArg.getNextSibling();
    }
  }
;

```

```

    }

    int count = repVector.size();
    repArray = new AbstractRepresentation[count];
    for(int i = 0; i < count; i++)
        repArray[i] = (AbstractRepresentation) repVector.elementAt(i);
}
;

expression returns [AbstractRepresentation exprRep]
{
    String o = null;
    AbstractRepresentation a = null, b = null, c = null;
    SandboxAST streamNode = null;
    SandboxAST tempNode = null;
    SandboxAST callName = null;
    exprRep = null;
    debugStream.println("    evaluating expression: " + _t.toStringList());
}
: callNode:CALL
  {
    streamNode = (SandboxAST) callNode.getFirstChild(); // should be name of stream, or name of function called.

    String typeName = getType(streamNode.getText());

    // Handle method calls on ofstream and ifstream objects.
    if (typeName != null && (typeName.equals("ofstream") || typeName.equals("ifstream"))) {

        // Reads and ignores the 'dot' in the tree
        tempNode = (SandboxAST) streamNode.getNextSibling();

        // Reads the name of the method call
        callName = (SandboxAST) tempNode.getNextSibling();

        // get the arguments list
        SandboxAST argsList = (SandboxAST) callName.getNextSibling();

        // Attempt to find the function in a library and then invoke it if found.
        // Currently this will only work for methods without parameters (specifically, close()).
        // Will need to be expanded when we get to open, ignore, setw, etc.
        Class sRepClass = null;
        String funcName = callName.getText();
        AbstractRepresentation sRep = symtab.getRepresentation(streamNode.getText());
        Class[] actTypes = getActualTypes((SandboxAST) argsList);
        Object[] actValues = getActualValues((SandboxAST) argsList);

        try {
            Object returnVal;

            if(typeName.equals("ofstream")) {
                OFStreamRepresentation osRep = (OFStreamRepresentation) sRep;
                sRepClass = OFStreamRepresentation.class;
                Method methodToCall = sRepClass.getMethod(funcName, actTypes);
                returnVal = methodToCall.invoke(osRep, actValues);
            }
            else {
                IFStreamRepresentation isRep = (IFStreamRepresentation) sRep;
                sRepClass = IFStreamRepresentation.class;
                Method methodToCall = sRepClass.getMethod(funcName, actTypes);
                returnVal = methodToCall.invoke(isRep, actValues);
            }

            if(returnVal != null)
            {
                if(returnVal instanceof Boolean)
                    exprRep = new BoolRepresentation( ((Boolean) returnVal).booleanValue() );
                else if(returnVal instanceof Character)
                    exprRep = new CharRepresentation( ((Character) returnVal).charValue() );
            }
        }
        catch(NoSuchMethodException nsme) {
            debugStream.println("    no such method exception.");
        }
    }
}

```

```

// OK, we didn't find an exact match for the intended function call,
// but we know a suitable one exists if it got through the semantic checker...
// now we just need to jump through the hoops again to find it!
Method[] streamMethods = sRepClass.getMethods();
Class[] streamParamTypes = null;
int bestFitIndex = -1;
int leastConversions = 999;
for (int index=0; index < streamMethods.length; index++) {
    // Since we get ALL public methods, we need to check both the names and the paramlist size
    if (streamMethods[index].getName().equals(funcName) &&
        streamMethods[index].getParameterTypes().length == actTypes.length) {
        // Start worrying finding each possible matching function here
        streamParamTypes = streamMethods[index].getParameterTypes();
        int sumConversions = 0;
        for (int paramIndex = 0; paramIndex < streamParamTypes.length; paramIndex++)
            sumConversions += typeManager.checkConversionSteps(
                NewSemantic.convertClassToAbstract(streamParamTypes[paramIndex]),
                ((AbstractRepresentation) actValues[paramIndex]).getType());

        // Keep a reference to the single best match found
        if (sumConversions > 0 && sumConversions < leastConversions) {
            leastConversions = sumConversions;
            bestFitIndex = index;
        }
    }
}
if (bestFitIndex != -1) {
    // If we've found the match, run with it!
    Method streamMethod = streamMethods[bestFitIndex];
    try {
        System.out.println("found it.");
        streamMethod.invoke(null,
            getConvertedActualValues((SandboxAST) streamNode.getNextSibling(), streamParamTypes));
    } catch (Exception e) {
        debugStream.println("...exception caught during invocation of function.");
        System.err.println("Exception caught during invocation of function.");
        e.printStackTrace();
    }
}
}
catch(InvocationTargetException e) {
    debugStream.println("...exception caught during invocation of function.");
    e.printStackTrace();
}
catch(Exception e) {
    // This IS something to worry about; shouldn't get here.
    e.printStackTrace();
}
} else if (typeName != null && typeName.equals("string")) {
    // Handle method calls on string objects.

    // Reads and ignores the 'dot' in the tree
    tempNode = (SandboxAST) streamNode.getNextSibling();

    // Reads the name of the method call
    callName = (SandboxAST) tempNode.getNextSibling();

    // Reads the arguments list
    SandboxAST argsList = (SandboxAST) callName.getNextSibling();

    // Attempt to find the function in a library and then invoke it if found.
    try {
        StringRepresentation stringRepresentation = (StringRepresentation)
            symtab.getRepresentation(streamNode.getText());
        Method methodToCall = StringRepresentation.class.getDeclaredMethod(callName.getText(), new Class[] { });
        Object valueReturned = methodToCall.invoke(stringRepresentation, new Object[] { });
        exprRep = new IntRepresentation(((Integer) valueReturned).intValue());
    }
    catch(NoSuchMethodException e) {
        // This would be bad...
        e.printStackTrace( System.out );
    }
}
catch(Exception e) {
    // This IS something to worry about; shouldn't get here.
}

```

```

        e.printStackTrace( System.out );
    }
} else if (typeName != null && (typeName.equals("istream"))) {
    // Handle method calls on istream objects.

    // Reads and ignores the 'dot' in the tree
    tempNode = (SandboxAST) streamNode.getNextSibling();

    // Reads the name of the method call
    callName = (SandboxAST) tempNode.getNextSibling();

    // Reads the arguments list
    SandboxAST argsList = (SandboxAST) callName.getNextSibling();

    // retrieve the stream to use.
    IStreamRepresentation streamRepresentation =
        (IStreamRepresentation) symtab.getRepresentation(streamNode.getText());

    try
    {
        Class sRepClass = IStreamRepresentation.class;
        String funcName = callName.getText();
        Class[] actTypes = getActualTypes((SandboxAST) argsList);
        Object[] actValues = getActualValues((SandboxAST) argsList);
        try
        {
            Method methodToCall = sRepClass.getMethod(funcName, actTypes);
            methodToCall.invoke(streamRepresentation, actValues);
        }
        catch( Exception e )
        {
            e.printStackTrace( System.out );
        }
    } catch (Exception io) {
        debugStream.println("...exception caught");
        System.err.println("IStream failed");
        io.printStackTrace();
    }
}

// Handle calls to user-defined functions first (if possible), and then libraries.
} else {
    String funcName = streamNode.getText();
    tempNode = (SandboxAST) streamNode.getNextSibling(); // gets the "ARGUMENTS" node
    AbstractRepresentation[] repArray = null;

    FunctionWrapper function = null;
    if (tempNode != null) {
        repArray = arguments(tempNode);
        function = fm.locateFunction(funcName, repArray.length);
    } else
        function = fm.locateFunction(funcName, 0);

    if (function != null) {
        if (function.getFunctionRoot() != null) {
            // The call is to a user-defined function
            debugStream.println("Calling user-defined function: " + funcName);
            exprRep = function.invoke(this, repArray);
        } else {
            reportError(funcName, ErrorCodeTable.UNABLE_TO_LOCATE_FUNCTION, ((SandboxAST) streamNode).getLine(), 0);
            debugStream.println(funcName + " function not found, possible scope issue...");
            // Don't reset exprRep so that a null is returned. This is akin to a linker error,
            // but we don't want anything
            // meaningful to happen... may result in an internal exception being thrown, but that's ok...
        }
    }
} else if (libraryManager.containsFunction(funcName)) {
    // The call is to a library function
    debugStream.println("Calling library function: " + funcName);

    // Ensure what's in the library is call-able, otherwise, it's likely to be
    // handled here in the interpreter, above, as a special function (for I/O)!
    if (libraryManager.inLibrary(funcName)) {
        debugStream.println(" " + funcName + " found in library file.");
        Class libClass = null;
        Class[] actTypes = getActualTypes((SandboxAST) streamNode.getNextSibling());
    }
}

```

```

try {
    libClass = Class.forName(libraryManager.getOwner(funcName));
    Method libMethod = libClass.getMethod(funcName, actTypes);
    SandboxAST paramStartNode = (SandboxAST) streamNode.getNextSibling();
    Object args[] = getActualValues(paramStartNode);
    Object valueReturned = libMethod.invoke(null, args);

    String returnType = libMethod.getReturnType().getName();
    // add CHARS here and test!
    if (returnType.equals("double")) {
        exprRep = new DoubleRepresentation(((Double) valueReturned).doubleValue());
        if (Double.isNaN(((DoubleRepresentation) exprRep).getValue())) {
            reportError(null, ErrorCodeTable.NOT_A_NUMBER, ((SandboxAST)
                callNode.getFirstChild()).getLine(), 0);
            debugStream.println("  Run time error: expression value is not a number.");
        }
    }

    } else if (returnType.equals("int"))
        exprRep = new IntRepresentation(((Integer) valueReturned).intValue());

    else if (returnType.equals("boolean"))
        exprRep = new BoolRepresentation(((Boolean) valueReturned).booleanValue());

    else if (returnType.equals("void"))
        exprRep = null;

    if (libMethod.getName().equals("getline")) {
        // Replace the string destination (2nd formal parameter) with the returned value
        SandboxAST paramNode = (SandboxAST) streamNode.getNextSibling().getFirstChild().getNextSibling();
        StringRepresentation rep = (StringRepresentation) symtab.getRepresentation(paramNode.getText());
        rep.setValue(((StringRepresentation) args[1]).getValue());
        exprRep = null; // Force function to return a null, since we overrode it with String
        //to get modified value out...
    }
} catch (NoSuchMethodException nsme) {
    debugStream.println("  no such method exception.");

    // OK, we didn't find an exact match for the intended function call,
    // but we know a suitable one exists if it got through the semantic checker...
    // now we just need to jump through the hoops again to find it!
    Method[] libMethods = libClass.getMethods();
    Class[] libParamTypes = null;
    int bestFitIndex = -1;
    int leastConversions = 999;
    for (int index=0; index < libMethods.length; index++) {
        // Since we get ALL public methods, we need to check both the names and the paramlist size
        if (libMethods[index].getName().equals(funcName) &&
            libMethods[index].getParameterTypes().length == actTypes.length) {
            // Start worrying finding each possible matching function here
            libParamTypes = libMethods[index].getParameterTypes();
            int sumConversions = 0;
            for (int paramIndex = 0; paramIndex < libParamTypes.length; paramIndex++)
                sumConversions += typeManager.checkConversionSteps(
                    NewSemantic.convertClassToAbstract(libParamTypes[paramIndex]),
                    NewSemantic.convertClassToAbstract(actTypes[paramIndex]));

            // Keep a reference to the single best match found
            if (sumConversions > 0 && sumConversions < leastConversions) {
                leastConversions = sumConversions;
                bestFitIndex = index;
            }
        }
    }
}
if (bestFitIndex != -1) {
    // If we've found the match, run with it!
    Method libMethod = libMethods[bestFitIndex];
    try {
        Object valueReturned = libMethod.invoke(null,
            getConvertedActualValues((SandboxAST) streamNode.getNextSibling(), libParamTypes));

        String returnType = libMethod.getReturnType().getName();
        // add CHARS here and test!
    }
}

```



```

    b = evaluateBinaryOp(MINUS, a, b);
    AST leftNode = mAssignNode.getFirstChild();
    RepresentationRecord rr = syntab.getRepresentationRecord(leftNode.getText());

    if (rr != null && rr.getRepresentation().isConstant()) {
        reportError(leftNode.getText() + " is a constant; it can not be modified.",
            ErrorCodeTable.RUNTIME_ASSIGNMENT_TO_CONSTANT, ((SandboxAST) leftNode).getLine(), 0);
        debugStream.println(" Run time error: attempting to modify a constant.");
    } else
        exprRep = evaluateAssignment(MINUS_ASSIGN, a, b);
}
| #(sAssignNode:STAR_ASSIGN a=expression b=expression)
{
    b = evaluateBinaryOp(STAR, a, b);
    AST leftNode = sAssignNode.getFirstChild();
    RepresentationRecord rr = syntab.getRepresentationRecord(leftNode.getText());

    if (rr != null && rr.getRepresentation().isConstant()) {
        reportError(leftNode.getText() + " is a constant; it can not be modified.",
            ErrorCodeTable.RUNTIME_ASSIGNMENT_TO_CONSTANT, ((SandboxAST) leftNode).getLine(), 0);
        debugStream.println(" Run time error: attempting to modify a constant.");
    } else
        exprRep = evaluateAssignment(STAR_ASSIGN, a, b);
}
| #(slAssignNode:SLASH_ASSIGN a=expression b=expression)
{
    b = evaluateBinaryOp(SLASH, a, b);
    AST leftNode = slAssignNode.getFirstChild();
    RepresentationRecord rr = syntab.getRepresentationRecord(leftNode.getText());

    if (rr != null && rr.getRepresentation().isConstant()) {
        reportError(leftNode.getText() + " is a constant; it can not be modified.",
            ErrorCodeTable.RUNTIME_ASSIGNMENT_TO_CONSTANT, ((SandboxAST) leftNode).getLine(), 0);
        debugStream.println(" Run time error: attempting to modify a constant.");
    } else
        exprRep = evaluateAssignment(SLASH_ASSIGN, a, b);
}
| #(perAssignNode:PERCENT_ASSIGN a=expression b=expression)
{
    b = evaluateBinaryOp(PERCENT, a, b);
    AST leftNode = perAssignNode.getFirstChild();
    RepresentationRecord rr = syntab.getRepresentationRecord(leftNode.getText());

    if (rr != null && rr.getRepresentation().isConstant()) {
        reportError(leftNode.getText() + " is a constant; it can not be modified.",
            ErrorCodeTable.RUNTIME_ASSIGNMENT_TO_CONSTANT, ((SandboxAST) leftNode).getLine(), 0);
        debugStream.println(" Run time error: attempting to modify a constant.");
    } else
        exprRep = evaluateAssignment(PERCENT_ASSIGN, a, b);
}
| #(EQ a=expression b=expression)
{
    // Check for stream types
    if ((a != null) && (b != null)) {
        // The stream MUST be compared to a bool. Conversion is not necessary
        if (a.getType().isStream() && (b instanceof BoolRepresentation)) {
            boolean isFailed = ((Stream)a).getFailed();
            boolean boolValue = ((BoolRepresentation)b).getValue();
            exprRep = new BoolRepresentation(!isFailed == boolValue);
        }
        else if (b.getType().isStream() && (a instanceof BoolRepresentation)) {
            boolean isFailed = ((Stream)b).getFailed();
            boolean boolValue = ((BoolRepresentation)a).getValue();
            exprRep = new BoolRepresentation(!isFailed == boolValue);
        }
        else
            exprRep = evaluateBinaryOp(EQ, a, b);
    }
    else
        exprRep = evaluateBinaryOp(EQ, a, b);
}
| #(NE a=expression b=expression)
{
    // Check for stream types

```

```

    if ((a != null) && (b != null)) {
        // The stream MUST be compared to a bool. Conversion is not necessary
        if (a.getType().isStream() && (b instanceof BoolRepresentation)) {
            boolean isFailed = ((Stream)a).getFailed();
            boolean boolValue= ((BoolRepresentation)b).getValue();
            exprRep= new BoolRepresentation(!isFailed != boolValue);
        }
        else if (b.getType().isStream() && (a instanceof BoolRepresentation)) {
            boolean isFailed = ((Stream)b).getFailed();
            boolean boolValue= ((BoolRepresentation)a).getValue();
            exprRep= new BoolRepresentation(!isFailed != boolValue);
        }
        else
            exprRep = evaluateBinaryOp(NE, a, b);
    }
    else
        exprRep = evaluateBinaryOp(NE, a, b);
}
| #(LT a=expression b=expression)
{
    exprRep = evaluateBinaryOp(LT, a, b);
}
| #(GT a=expression b=expression)
{
    exprRep = evaluateBinaryOp(GT, a, b);
}
| #(LE a=expression b=expression)
{
    exprRep = evaluateBinaryOp(LE, a, b);
}
| #(GE a=expression b=expression)
{
    exprRep = evaluateBinaryOp(GE, a, b);
}
| #(POST_INCR a=expression)
{
    b = evaluateBinaryOp(PLUS, a, ONE);
    exprRep = a.cloneRepresentation();
    evaluateAssignment(ASSIGN, a, b);
}
| #(POST_DECR a=expression)
{
    b = evaluateBinaryOp(MINUS, a, ONE);
    exprRep = a.cloneRepresentation();
    evaluateAssignment(ASSIGN, a, b);
}
//PREFIX increment...
| #(INCR a=expression)
{
    b = evaluateBinaryOp(PLUS, a, ONE);
    exprRep = evaluateAssignment(ASSIGN, a, b);
    exprRep = a;
}
| #(DECR a=expression)
{
    b = evaluateBinaryOp(MINUS, a, ONE);
    exprRep = evaluateAssignment(ASSIGN, a, b);
    exprRep = a;
}
| #(PLUS a=expression b=expression)
{
    exprRep = evaluateBinaryOp(PLUS, a, b);
}
| #(MINUS a=expression (b=expression)?)
{
    exprRep = evaluateBinaryOp(MINUS, a, b);
}
| #(STAR a=expression b=expression)
{
    exprRep = evaluateBinaryOp(STAR, a, b);
}
| #(sl:SLASH a=expression b=expression)
{

```

```

        currentLineNumber = ((SandboxAST) sl).getLine();
        exprRep = evaluateBinaryOp(SLASH, a, b);
    }
| #(pct:PERCENT a=expression b=expression)
{
    currentLineNumber = ((SandboxAST) pct).getLine();
    exprRep = evaluateBinaryOp(PERCENT, a, b);
}
| #(lshiftNode:LSHIFT a=expression b=expression)
{
    // Handling the case of calling a stream method...
    if (a != null && (a.getType().getName().equals("ofstream") || (a.getType().getName().equals("ostream")))
        && b != null && b.getType().getName().equals("string")) {

        StringRepresentation bstrRep = (StringRepresentation) b;
        // Handle calling a setprecision on a file or output stream

        if (bstrRep.getValue() != null)
        {
            String manipName = bstrRep.getValue();
            OStreamRepresentation osRep = (OStreamRepresentation) a;

            AST callNode2 = lshiftNode.getFirstChild(); // gets the ofstream node
            callNode2 = callNode2.getNextSibling(); // gets the next node (CALL or identifier)

            if (callNode2.getText().equals("CALL"))
            {
                callNode2 = callNode2.getFirstChild(); // gets the manip node
                callNode2 = callNode2.getNextSibling(); // gets the ARGUMENTS and parameter nodes

                Class[] actTypes = getActualTypes( (SandboxAST) callNode2 );
                Object[] actVals = getActualValues( (SandboxAST) callNode2 );

                try {
                    Method methodToCall = OStreamRepresentation.class.getMethod(manipName, actTypes);
                    methodToCall.invoke(osRep, actVals);
                    exprRep = osRep;
                }
                catch(NoSuchMethodException e) {
                    // No problem; wasn't a setprecision, setw, or setfill call. Just carry on.
                    exprRep = evaluateBinaryOp(LSHIFT, a, b);
                }
                catch(Exception e) { /* This won't happen. */ }
            }
            else
                exprRep = evaluateBinaryOp(LSHIFT, a, b);
        }
    }
    else
        exprRep = evaluateBinaryOp(LSHIFT, a, b);
}
| #(RSHIFT a=expression b=expression)
{
    exprRep = evaluateBinaryOp(RSHIFT, a, b);
    if (a.getType().getName().equals("istream") || a.getType().getName().equals("ifstream"))
        // Return the "correct" result, though we converted to another data type...
        // this permits following expressions (if any) to utilize i*streams in the expression
        exprRep = a;
}
| exprRep=primaryExpr
| bangNode:BANG
{
    boolean result = evaluateBoolExpression(bangNode.getFirstChild());

    debugStream.println(" BANG operator, turning " + result + " into " + !result);
    exprRep = new BoolRepresentation(!result);
}
| landNode:LAND
{
    debugStream.println(" evaluating &&");

    AST node = landNode.getFirstChild();
    boolean value = evaluateBoolExpression(node);
    if (value)

```

```
        value = evaluateBoolExpression(node.getNextSibling());
    else
        debugStream.println("  evaluation short-circuited");
        exprRep = new BoolRepresentation(value);
    }
| lorNode:LOR
  {
    debugStream.println("evaluating ||");

    AST node = lorNode.getFirstChild();
    boolean value = evaluateBoolExpression(node);

    if (!value)
        value = evaluateBoolExpression(node.getNextSibling());
    else
        debugStream.println("  evaluation short-circuited");
        exprRep = new BoolRepresentation(value);
  }
;
```

J.8 ANTLR-produced Output File for the *CS1 Sandbox's* Interpreter (Interp.java)

```
// $ANTLR 2.7.1: "NewInterp.g" -> "NewInterp.java"$

package sandbox.compiler.backend;

import java.io.*;
import java.util.*;
import java.lang.reflect.*;
import sandbox.client.*;
import sandbox.compiler.*;
import sandbox.compiler.type.*;
import sandbox.compiler.symtab.*;
import sandbox.compiler.frontend.*;
import sandbox.lib.*;
import sandbox.lib.string.*;
import sandbox.lib.iostream.*;

import antlr.TreeParser;
import antlr.Token;
import antlr.collections.AST;
import antlr.RecognitionException;
import antlr.ANTLRException;
import antlr.NoViableAltException;
import antlr.MismatchedTokenException;
import antlr.SemanticException;
import antlr.collections.impl.BitSet;
import antlr.ASTPair;
import antlr.collections.impl.ASTArray;

/**
 * NewInterp is the last step in the interpretation process.
 * After having gone through the parser, and passed the semantic check,
 * we are now ready to execute the program.
 *
 * $Id: interpSource.tex,v 1.2 2003/07/01 18:21:31 pete Exp $
 */
public class NewInterp extends antlr.TreeParser
    implements NewInterpTokenTypes
{
    // Items related to the debugging stream
    //
    // If you want to put an error message to the debug stream
    // do something like this: debugStream.println("This is my error.");
    ByteArrayOutputStream out = new ByteArrayOutputStream();
    PrintStream debugStream = new PrintStream(out);

    // Create the symbol table as a "global" variable.
    SymTab symtab = new LinkSymTab();

    private LibraryManager libraryManager = null;
    private TypeManager typeManager = null;
    private AbstractRepresentation ONE = new IntRepresentation(1);
    private MenuButtonListener threadKiller = null;
    private IOFrame iof = null;

    // The function manager, manages the addition, use, and query of user-defined functions
    private FunctionManager fm;

    // Holds the wrapper to the function under execution.
    // private FunctionWrapper currentFunction = null;
    private Stack functionStack = new Stack();

    // Used to track current line numbers for run-time errors (when direct access to the AST node
    // is not possible and passing the node means changing a lot of code)
    private int currentLineNumber = 0;
}

```

```
* Constructor which accepts a reference to the function manager.
*/
public NewInterp (FunctionManager funcMgr) {
    fm = funcMgr;
}

/**
 * Sets the library manager to the parameter, also sets the
 * reference to the type manager.
 */
public void setLibraryManager(LibraryManager mgr) {
    libraryManager = mgr;
    typeManager = libraryManager.getTypeManager();
}

/**
 * Registers the IOFrame used for standard input (cin) and standard output (cout)
 */
public void setIOFrame(IOFrame iof) {
    this.iof = iof;
}

/** The errors that are observed during the parse phase. */
private Vector errorVector = new Vector();

/**
 * Returns a vector of each error seen during compilation.
 */
public Vector getErrorVector() {
    return(errorVector);
}

public void reportError (int code, int line, int col) {
    reportError("", code, line, col);
}

public void reportError (java.lang.String errMsg, int code, int line, int col) {
    AntlrError e = new AntlrError ();
    e.setType("Interpreter error");
    e.setErrorMessage(errMsg);
    e.setErrorCode(code);
    e.setLineNumber(line);
    e.setColumnNumber(col);
    errorVector.add(e);
}

public void reportError(RecognitionException ex) {
    AntlrError error = new AntlrError();
    error.setType("recognition exception -- interpreter");
    error.setErrorMessage(ex.getMessage());
    error.setLineNumber(ex.getLine());
    error.setColumnNumber(ex.getColumn());
    errorVector.add(error);
}

public void reportError(java.lang.String s) {
    AntlrError error = new AntlrError();
    error.setType("error");
    error.setErrorMessage(s);
    errorVector.add(error);
}

public void reportWarning(java.lang.String s) {
    AntlrError error = new AntlrError();
    error.setType("warning");
}
```

```

    error.setErrorMessage(s);
    errorVector.add(error);
}

/**
 * Sets this interpreter's debugging PrintStream object
 */
public void setDebugOutput(PrintStream p) {
    debugStream = p;
}

/**
 * Returns the OutputStream for this interpreter
 */
public java.lang.String getDebugOutput() {
    return(out.toString());
}

/**
 * Returns the string name of the type for identifier 'identifName', assuming it exists.
 */
private java.lang.String getType(java.lang.String identifName) {
    java.lang.String returnType = null;
    if (symtab == null) System.err.println("Symtab is null in interp.");

    if ((symtab != null) && (identifName != null)) {
        AbstractRepresentation abRep = symtab.getRepresentation(identifName);
        if (abRep != null)
            returnType = abRep.getType().getName();
    }
    return returnType;
}

/**
 * Builds a new type map for the primitive types and ensures they are placed in the
 * symbol table.
 */
protected void assessTypes() {
    typeManager.buildTypeMap();
    for(Enumeration e = typeManager.getTypeNames(); e.hasMoreElements(); ) {
        java.lang.String typeName = (java.lang.String) e.nextElement();
        AbstractType type = typeManager.getType(typeName);
        TypeRepresentation rep = new TypeRepresentation(type);

        symtab.setRepresentationRecord(typeName, new RepresentationRecord());
        symtab.setRepresentation(typeName, rep);
    }
}

/**
 * Evaluates a boolean expression (!, &&, ||) and returns the appropriate result.
 */
protected boolean evaluateBoolExpression(AST node) throws RecognitionException {
    AbstractRepresentation rep = expression(node);
    try {
        // Try to convert to a boolean
        BoolRepresentation bRep = (BoolRepresentation) typeManager.convert(rep, new BoolType());

        return bRep.getValue();
    } catch(Exception ex) {
        if (rep.getType().getName().equals("istream") || rep.getType().getName().equals("ifstream")) {
            return (!((IStreamRepresentation) rep).getFailed();
        } else if (rep.getType().getName().equals("ostream") || rep.getType().getName().equals("ofstream")) {
            return (!((OStreamRepresentation) rep).getFailed();
        } else
            return false;
    }
}

```

```

/**
 * Evaluates a given expression (generally a test expression used in for loops, while loops,
 * and if statements) and returns a boolean true or false value.
 */
protected boolean evaluateTestExpression(AST node) throws RecognitionException {
    AbstractRepresentation rep = expression(node);
    try {
        // Is the representation a stream??
        AbstractType type = rep.getType();
        if (type.isStream()) {
            // try to convert to a stream object and return if the stream has failed.
            Stream stream = (Stream) rep;
            return (!stream.getFailed());
        }
        else { // the value should be numeric
            // Try to convert to an integer
            IntRepresentation iRep = (IntRepresentation)typeManager.convert(rep, new IntType());
            return (iRep.getValue() != 0);
        }
    } catch(Exception ex) {
        ex.printStackTrace();
    }

    return false;
}

/**
 * Attempts to evaluate a binary operation expression given the operator ('+', '-', '*', etc.)
 * and the left-hand side (lhs) and the right-hand side (rhs). Really passes this to the type
 * manager for execution (interpretation).
 */
protected AbstractRepresentation evaluateBinaryOp(int operator, AbstractRepresentation lhs, AbstractRepresentation rhs) {
    try {

        debugStream.print("  binary op " + operator + " on " + lhs.getType().getName() + " and " + rhs.getType().getName());
        AbstractRepresentation rep = typeManager.evaluateBinaryOp(operator, lhs, rhs);
        if (rep != null)
            debugStream.println("...result is " + rep.getType().getName());

        return rep;
    } catch(Exception ex) {
        // The following section catches a negative assignment and returns the correct value
        // and also catches the divide by zero problem and sets a run-time error message.
        if (operator == MINUS) {
            try {
                rhs.getType();
            } catch(Exception ex2) {
                return lhs.negate(); // negates the current value - Dave 2/14/02
            }
        } else if ((operator == SLASH || operator == PERCENT) && ex.getMessage().equals("/ by zero")) {
            reportError(null, ErrorCodeTable.DIVIDE_BY_ZERO, currentLineNumber, 0);
            debugStream.println("  Run time error: divide by zero.");
            return null;
        }
        debugStream.println("...exception caught");
        ex.printStackTrace();
        return null;
    }
}

/**
 * Attempts to evaluate an assignment expression given the operator (remember that assignments
 * are not just '=', but '+=', '-=', etc.) and the left-hand side (lhs) and the right-hand
 * side (rhs). An implicit type conversion (type coercion) takes place first (if needed).
 */
protected AbstractRepresentation evaluateAssignment(int op, AbstractRepresentation lhs, AbstractRepresentation rhs) {
    try {

        rhs = typeManager.convert(rhs, lhs.getType());
        lhs.setValue(rhs);
    } catch(Exception ex) {
        ex.printStackTrace();
    }
}

```

```

    }
    finally {
        return rhs;
    }
}

/**
 * Attempts to evaluate the passed in value by determining if it is a constant
 * (character constant [e.g., '.'], string literal [e.g., "hi"], etc..). If the
 * variable is not a constant then it must be an identifier. The value of the
 * identifier is then determined and returned.
 */
protected AbstractRepresentation evaluateVariable(SandboxAST value, AbstractType type) throws IncompatibleTypesException {
    AbstractRepresentation rep= null; // The representation of the value
    try {
        // test if the value is a literal
        rep = constant(value);
    } catch (RecognitionException re) {
        System.err.println("Recognition exception in evaluateVariable");
        re.printStackTrace();
    }

    if (rep == null) { // it is not a constant, it must be an identifier
        rep = symtab.getRepresentation(value.getText());
    }

    return typeManager.convert(rep, type);
}

/**
 * Sets a reference to the MenuButtonListener so we can check safely for an interruption
 * of this thread (in the event of endless loops!)
 */
public void setListener(MenuButtonListener mbl) {
    threadKiller = mbl;
}

/**
 * Returns an array of class objects of actual parameter types to a library function call.
 * ctuals is assumed to be the ARGUMENTS node in the AST, so we first need to skip down
 * to the first parameter.
 */
protected Class[] getActualTypes(SandboxAST actuals) {
    if(actuals == null)
        return new Class[] {};

    SandboxAST param = (SandboxAST) actuals.getFirstChild();
    ArrayList tempList = new ArrayList();

    while (param != null) {
        try {
            AbstractRepresentation rep = expression((AST) param);
            if (rep instanceof DoubleRepresentation || rep instanceof FloatRepresentation)
                tempList.add(DoubleRepresentation.class);

            else if (rep instanceof IntRepresentation || rep instanceof LongRepresentation ||
                    rep instanceof ShortRepresentation)
                tempList.add(IntRepresentation.class);

            else if (rep instanceof BoolRepresentation)
                tempList.add(BoolRepresentation.class);

            else if (rep instanceof CharRepresentation)
                tempList.add(CharRepresentation.class);

            else if (rep instanceof IStreamRepresentation || rep instanceof IFStreamRepresentation)
                tempList.add(IStreamRepresentation.class);

            else if (rep instanceof StringRepresentation)
                tempList.add(sandbox.lib.string.StringRepresentation.class);
        }
    }
}

```

```

        else if (rep instanceof StringLiteralRepresentation)
            tempList.add(StringLiteralRepresentation.class);

        param = (SandboxAST) param.getNextSibling();
    } catch (RecognitionException re) {
        re.printStackTrace();
    }
}

// Build and return the list of class objects
int count = tempList.size();
Class[] classArray = new Class[count];
for (int i=0; i < count; i++)
    classArray[i] = (Class) tempList.get(i);

return classArray;
}

/**
 * Returns an array of objects of the values of the actual parameters to a library function call.
 * Primitive data type parameters must be wrapped in wrapper classes to utilize Java
 * reflection, needed to call the library function. Actuals is assumed to be the ARGUMENTS
 * node in the AST, so we first need to skip down to the first parameter.
 */
protected Object[] getActualValues(SandboxAST actuals) {
    if(actuals == null)
        return new Object[] {};

    SandboxAST param = (SandboxAST) actuals.getFirstChild();
    ArrayList tempList = new ArrayList();

    while (param != null) {
        try {
            AbstractRepresentation rep = expression((AST) param);
/*
            if (rep instanceof DoubleRepresentation)
                tempList.add(new Double(((DoubleRepresentation) rep).getValue()));

            else if (rep instanceof FloatRepresentation)
                tempList.add(new Double(((FloatRepresentation) rep).getValue()));

            else if (rep instanceof LongRepresentation)
                tempList.add(new Integer((int)((LongRepresentation) rep).getValue()));

            else if (rep instanceof IntRepresentation)
                tempList.add(new Integer(((IntRepresentation) rep).getValue()));

            else if (rep instanceof BoolRepresentation)
                tempList.add(new Boolean(((BoolRepresentation) rep).getValue()));

            else if (rep instanceof CharRepresentation)
                tempList.add((CharRepresentation) rep);

            else if ((rep instanceof IStreamRepresentation) || (rep instanceof IFStreamRepresentation))
                tempList.add((IStreamRepresentation) rep);

            else if (rep instanceof StringLiteralRepresentation)
                tempList.add((StringLiteralRepresentation) rep);
*/

            if (rep instanceof StringRepresentation)
                if (((StringRepresentation) rep).getValue() == null)
                    tempList.add(new StringRepresentation(""));
                else
                    tempList.add((StringRepresentation) rep);
            else if (rep instanceof LongRepresentation)
                tempList.add(new IntegerRepresentation(((LongRepresentation) rep).getValue()));
            else if (rep instanceof ShortRepresentation)
                tempList.add(new IntegerRepresentation(((ShortRepresentation) rep).getValue()));
            else
                tempList.add( rep );

            param = (SandboxAST) param.getNextSibling();

```

```

    } catch (RecognitionException re) {
        re.printStackTrace();
    }
}

// Build and return the list of objects (wrappers)
int count = tempList.size();
Object[] actualsArray = new Object[count];
for (int i=0; i < count; i++)
    actualsArray[i] = tempList.get(i);

return actualsArray;
}

/**
 * Returns an array of objects of the values of the actual parameters to a library function call.
 * Primitive data type parameters must be wrapped in wrapper classes to utilize Java
 * reflection, needed to call the library function. Actuals is assumed to be the ARGUMENTS
 * node in the AST, so we first need to skip down to the first parameter.
 * This method differs from the one above as it performs the coercion as it proceeds.
 */
protected Object[] getConvertedActualValues(SandboxAST actuals, Class[] formals) {
    SandboxAST param = (SandboxAST) actuals.getFirstChild();
    ArrayList tempList = new ArrayList();
    int index = 0;

    while (param != null) {
        try {
            AbstractRepresentation rep = expression((AST) param);

            if (formals[index].getName().equals("java.lang.Double")) {
                DoubleRepresentation dRep = (DoubleRepresentation) typeManager.convert(rep, new DoubleType());
                tempList.add(new Double(dRep.getValue()));
            }
            else if (formals[index].getName().equals("java.lang.Integer")) {
                IntRepresentation iRep = (IntRepresentation) typeManager.convert(rep, new IntType());
                tempList.add(new Integer(iRep.getValue()));
            }
            else if (formals[index].getName().equals("java.lang.Boolean")) {
                BoolRepresentation bRep = (BoolRepresentation) typeManager.convert(rep, new BoolType());
                tempList.add(new Boolean(bRep.getValue()));
            }
            else if (formals[index].getName().equals("java.lang.Character")) {
                CharRepresentation cRep = (CharRepresentation) typeManager.convert(rep, new CharType());
                tempList.add(new Character((char) cRep.getValue()));
            }
            else if (formals[index].getName().equals("sandbox.compiler.type.DoubleRepresentation")) {
                DoubleRepresentation dRep = (DoubleRepresentation) typeManager.convert(rep, new DoubleType());
                tempList.add(dRep);
            }
            else if (formals[index].getName().equals("sandbox.compiler.type.IntRepresentation")) {
                IntRepresentation iRep = (IntRepresentation) typeManager.convert(rep, new IntType());
                tempList.add(iRep);
            }
            param = (SandboxAST) param.getNextSibling();
            index++;
        } catch (RecognitionException re) {
            re.printStackTrace();
        } catch (IncompatibleTypesException ite) {
            // this should not happen as well, if we have gotten to the interp...
            ite.printStackTrace();
        }
    }

    // Build and return the list of objects (wrappers)
    int count = tempList.size();
    Object[] actualsArray = new Object[count];
    for (int i=0; i < count; i++)
        actualsArray[i] = tempList.get(i);

    return actualsArray;
}

```

```

/**
 * Converts the src type (a Java class, as in Integer.class) to the corresponding
 * internal primitive type (IntType) for use in calculating type conversions when
 * looking up library function calls.
 */
public static AbstractRepresentation convertClassToAbstract(Class src) {
    if (src.getName().equals("java.lang.Integer"))
        return (new IntRepresentation());
    else if (src.getName().equals("java.lang.Double"))
        return new DoubleRepresentation();
    else if (src.getName().equals("java.lang.Character"))
        return new CharRepresentation();
    else if (src.getName().equals("java.lang.Boolean"))
        return new BoolRepresentation();
    else
        return null; // this should never occur!
}
}
public NewInterp() {
    tokenNames = _tokenNames;
}

public final void program(AST _t) throws RecognitionException {

    AST program_AST_in = (AST)_t;

    assessTypes();

    try { // for error handling
        AST __t2 = _t;
        AST tmp1_AST_in = (AST)_t;
        match(_t,PROGRAM);
        _t = _t.getFirstChild();
        {
        _loop4:
        do {
            if (_t==null) _t=ASTNULL;
            if ((_t.getType()==INCLUDE)) {
                include(_t);
                _t = _retTree;
            }
            else {
                break _loop4;
            }
        }

        } while (true);
        {
        {
        _loop6:
        do {
            if (_t==null) _t=ASTNULL;
            if ((_t.getType()==USING_DIRECTIVE)) {
                using(_t);
                _t = _retTree;
            }
            else {
                break _loop6;
            }
        }

        } while (true);
        {
        {
        _loop8:
        do {
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType()) {
            case DECLARATION:
            {
                declaration(_t);
                _t = _retTree;
                break;
            }
            case FUNCTION:
            {

```

```

        functDef(_t);
        _t = _retTree;
        break;
    }
    case FUNCTION_PROTOTYPE:
    {
        functionPrototype(_t);
        _t = _retTree;
        break;
    }
    default:
    {
        break _loop8;
    }
} while (true);
}
_t = _t2;
_t = _t.getNextSibling();

FunctionWrapper function = fm.locateFunction("main", 0);
AbstractRepresentation[] params = { new IntRepresentation(0) };

symtab.setRepresentationRecord("__return", new RepresentationRecord());
function.invoke(this, params);

}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
}

public final void include(AST _t) throws RecognitionException {

    AST include_AST_in = (AST)_t;
    AST includeNode = null;

    try { // for error handling
        includeNode = (AST)_t;
        match(_t,INCLUDE);
        _t = _t.getNextSibling();

        try {
            java.lang.String libName = includeNode.getFirstChild().getText().trim();
            Library lib = libraryManager.loadLibrary(libName);

            assessTypes();
            if (libName.equals("iostream"))
                lib.loadSymbols(symtab, iof);
            else
                lib.loadSymbols(symtab);
        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }

    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final void using(AST _t) throws RecognitionException {

    AST using_AST_in = (AST)_t;

    try { // for error handling
        AST tmp2_AST_in = (AST)_t;
        match(_t,USING_DIRECTIVE);
        _t = _t.getNextSibling();
    }
}

```

```

    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final void declaration(AST _t) throws RecognitionException {

    AST declaration_AST_in = (AST)_t;
    AST declNode = null;

    try {        // for error handling
        declNode = (AST)_t;
        match(_t,DECLARATION);
        _t = _t.getNextSibling();

        AST modifierNode = declNode.getFirstChild();
        AST typeNode;
        boolean isConst = false;

        if (modifierNode.getText().equals("TYPE_MODIFIER")) {
            if (modifierNode.getFirstChild().getText().equals("const"))
                isConst = true;

            typeNode = modifierNode.getNextSibling();
        } else {
            typeNode = modifierNode;
            modifierNode = null;
        }
        AST declaratorNode = typeNode.getNextSibling();

        while (declaratorNode != null) {
            declarator(declaratorNode, typeNode.getText(), isConst);
            declaratorNode = declaratorNode.getNextSibling();
        }

    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final void functDef(AST _t) throws RecognitionException {

    AST functDef_AST_in = (AST)_t;

    try {        // for error handling
        AST tmp3_AST_in = (AST)_t;
        match(_t,FUNCTION);
        _t = _t.getNextSibling();

        /**
         * There is nothing here since all functions are added to the
         * Function Manager now.... this is not needed. Functions are placed in the
         * manager in the semantic pass, so there's nothing here to do!
         */

    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final void functionPrototype(AST _t) throws RecognitionException {

    AST functionPrototype_AST_in = (AST)_t;

    try {        // for error handling
        AST tmp4_AST_in = (AST)_t;

```

```

match(_t,FUNCTION_PROTOTYPE);
_t = _t.getNextSibling();

/**
 * There is nothing here since all functions are added to the
 * Function Manager now... this is not needed. Function prototypes are placed in the
 * manager in the semantic pass, so there's nothing here to do!
 */

}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
}

public final AbstractRepresentation function(AST _t,
    AbstractRepresentation[] params
) throws RecognitionException {
    AbstractRepresentation retval;

    AST function_AST_in = (AST)_t;
    AST funcNode = null;

    // The parameter 'params' is the list of the actual parameters

    retval = null;

    try { // for error handling
        funcNode = (AST)_t;
        match(_t,FUNCTION);
        _t = _t.getNextSibling();

        symtab.beginFunction();

        AST type = funcNode.getFirstChild(); // references the node containing the return type
        AST name = type.getNextSibling(); // references the node containing the name of the function
        AST tempNode = name.getNextSibling(); // references the next node, this is either the
        // parameters or the body (block)
        AST paramsNode = null; // references the start of the formal parameters list
        AST block = null;
        int fParamCount = 0;

        debugStream.println("Interpreting function: " + name.getText());

        if (!tempNode.getText().equals("BLOCK_STATEMENT")) {
            paramsNode = name.getNextSibling();
            fParamCount = formalParameterList(paramsNode, params);
            block = paramsNode.getNextSibling();
        } else
            block = tempNode;

        // Set the currentFunction to the function wrapper
        // currentFunction = fm.locateFunction(name.getText(), fParamCount);
        functionStack.push(fm.locateFunction(name.getText(), fParamCount));

        java.lang.String exitCode = statement(block);
        if (exitCode != null && exitCode.equals("return"))
            retval = symtab.getRepresentation("__return");

        symtab.endFunction();

    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
    return retval;
}

public final int formalParameterList(AST _t,

```

```

    AbstractRepresentation[] actuals
) throws RecognitionException {
    int params;

    AST formalParameterList_AST_in = (AST)_t;
    AST paramsNode = null;

    // The parameter 'actuals' is the list of the actual parameters

    params = 0;

    try { // for error handling
        paramsNode = (AST)_t;
        match(_t,PARAMETER_LIST);
        _t = _t.getNextSibling();

        AST singleParam = paramsNode.getFirstChild();

        while (singleParam != null) {
            formalParameter(singleParam, actuals[params]);
            singleParam = singleParam.getNextSibling();
            params++;
        }
    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
    return params;
}

public final void formalParameter(AST _t,
    AbstractRepresentation actual
) throws RecognitionException {

    AST formalParameter_AST_in = (AST)_t;
    AST formalNode = null;

    // The parameter 'actual' is the representation of an actual parameter

    AST refNode = null;
    AbstractRepresentation rep;

    try { // for error handling
        formalNode = (AST)_t;
        match(_t,PARAMETER);
        _t = _t.getNextSibling();

        AST typeNode = formalNode.getFirstChild();
        if (typeNode.getText().equals("TYPE_MODIFIER"))
            typeNode = typeNode.getNextSibling();

        AST paramNode = typeNode.getNextSibling();
        if (paramNode != null && paramNode.getText().equals("&")) {
            refNode = paramNode;
            paramNode = paramNode.getNextSibling();
            rep = actual;
        } else
            rep = actual.cloneRepresentation();

        // Create the representation record for this entry in the symbol table.
        RepresentationRecord newRR = new RepresentationRecord();

        // If the actual is really a constant, set the formal to be a constant
        // unbeknownst to the user!
        if (actual.isConstant())
            newRR.setConstant(true);

        // Add this parameter to the symbol table.
        symtab.setRepresentationRecord(paramNode.getText(), newRR);
    }
}

```

```

        symtab.setRepresentation(paramNode.getText(), rep);
        debugStream.println(" Formal parameter " + paramNode.getText() + " set to " + rep);
    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final java.lang.String statement(AST _t) throws RecognitionException {
    java.lang.String exitCode;

    AST statement_AST_in = (AST)_t;

    if (threadKiller.getStopThread())
        return "killThread";
    exitCode = null;

    try { // for error handling
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType()) {
        case WHILE:
            {
                exitCode=whileStatement(_t);
                _t = _retTree;
                break;
            }
        case FOR:
            {
                exitCode=forStatement(_t);
                _t = _retTree;
                break;
            }
        case DO:
            {
                exitCode=doStatement(_t);
                _t = _retTree;
                break;
            }
        case IF:
            {
                exitCode=ifStatement(_t);
                _t = _retTree;
                break;
            }
        case SWITCH:
            {
                exitCode=switchStatement(_t);
                _t = _retTree;
                break;
            }
        case EXPRESSION_STATEMENT:
            {
                expressionStatement(_t);
                _t = _retTree;
                break;
            }
        case BLOCK_STATEMENT:
            {
                exitCode=blockStatement(_t);
                _t = _retTree;
                break;
            }
        case DECLARATION:
            {
                declaration(_t);
                _t = _retTree;
                break;
            }
        case RETURN:
            {

```



```

public final java.lang.String forStatement(AST _t) throws RecognitionException {
    java.lang.String exitCode;

    AST forStatement_AST_in = (AST)_t;
    AST forInit = null;
    AST declr = null;
    AST forCond = null;
    AST forIter = null;

    exitCode = null;

    try { // for error handling
        AST __t25 = _t;
        AST tmp5_AST_in = (AST)_t;
        match(_t, FOR);
        _t = _t.getFirstChild();
        {
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType() ) {
        case FOR_INIT:
        {
            forInit = (AST)_t;
            match(_t, FOR_INIT);
            _t = _t.getNextSibling();
            break;
        }
        case DECLARATION:
        {
            declr = (AST)_t;
            match(_t, DECLARATION);
            _t = _t.getNextSibling();
            break;
        }
        default:
        {
            throw new NoViableAltException(_t);
        }
        }
        }
        forCond = (AST)_t;
        match(_t, FOR_COND);
        _t = _t.getNextSibling();
        forIter = (AST)_t;
        match(_t, FOR_ITER);
        _t = _t.getNextSibling();
        _t = __t25;
        _t = _t.getNextSibling();

        AST bodyNode = null;
        if (forIter.getNextSibling() != null)
            bodyNode = forIter.getNextSibling();

        if (declr != null && declr.getText().equals("DECLARATION")) {
            if (forIter.getFirstChild() != null) {
                // Declaration(s) present, forIter not null
                for(declaration(declr); evaluateTestExpression(forCond.getFirstChild());
                    expression(forIter.getFirstChild())) {
                    if (bodyNode != null) {
                        exitCode = statement(bodyNode);
                        if (exitCode != null) {
                            if (exitCode.equals("return"))
                                break;
                            else if (exitCode.equals("break")) {
                                exitCode = null;
                                break;
                            } else if (exitCode.equals("continue"))
                                exitCode = null;
                        }
                    }
                }
            }
        } else if (forIter.getFirstChild() == null) {
            // Declaration(s) present, forIter null
            for(declaration(declr); evaluateTestExpression(forCond.getFirstChild()); ) {

```

```

        if (bodyNode != null) {
            exitCode = statement(bodyNode);
            if (exitCode != null) {
                if (exitCode.equals("return"))
                    break;
                else if (exitCode.equals("break")) {
                    exitCode = null;
                    break;
                } else if (exitCode.equals("continue"))
                    exitCode = null;
            }
        }
    }
} else {
    if (forInit.getFirstChild() != null) {
        if (forIter.getFirstChild() != null) {
            // Init not null, Iter not null
            for(expression(forInit.getFirstChild()); evaluateTestExpression(forCond.getFirstChild());
                expression(forIter.getFirstChild())) {
                if (bodyNode != null) {
                    exitCode = statement(bodyNode);
                    if (exitCode != null) {
                        if (exitCode.equals("return"))
                            break;
                        else if (exitCode.equals("break")) {
                            exitCode = null;
                            break;
                        } else if (exitCode.equals("continue"))
                            exitCode = null;
                    }
                }
            }
        }
    } else if (forIter.getFirstChild() == null) {
        // Init not null, Iter null
        for(expression(forInit.getFirstChild()); evaluateTestExpression(forCond.getFirstChild()); ) {
            if (bodyNode != null) {
                exitCode = statement(bodyNode);
                if (exitCode != null) {
                    if (exitCode.equals("return"))
                        break;
                    else if (exitCode.equals("break")) {
                        exitCode = null;
                        break;
                    } else if (exitCode.equals("continue"))
                        exitCode = null;
                }
            }
        }
    }
} else if (forInit.getFirstChild() == null) {
    if (forIter.getFirstChild() != null) {
        // Init null, Iter not null
        for( ; evaluateTestExpression(forCond.getFirstChild()); expression(forIter.getFirstChild())) {
            if (bodyNode != null) {
                exitCode = statement(bodyNode);
                if (exitCode != null) {
                    if (exitCode.equals("return"))
                        break;
                    else if (exitCode.equals("break")) {
                        exitCode = null;
                        break;
                    } else if (exitCode.equals("continue"))
                        exitCode = null;
                }
            }
        }
    }
} else if (forIter.getFirstChild() == null) {
    // Init null, Iter null
    for( ; evaluateTestExpression(forCond.getFirstChild()); ) {
        if (bodyNode != null) {
            exitCode = statement(bodyNode);
            if (exitCode != null) {
                if (exitCode.equals("return"))

```



```

try {      // for error handling
    ifNode = (AST)_t;
    match(_t,IF);
    _t = _t.getNextSibling();

    debugStream.println("Executing if statement");
    AST exprNode = ifNode.getFirstChild();
    AST trueNode = exprNode.getNextSibling();
    AST falseNode = trueNode.getNextSibling();
    boolean result = evaluateTestExpression(exprNode);

    if (result) {
        debugStream.println(" Following the true branch:");
        exitCode = statement(trueNode);
    } else if (falseNode != null) {
        debugStream.println(" Following the false branch:");
        exitCode=statement(falseNode);
    } else {
        debugStream.println(" Following the false, but no branch exists.");
        exitCode = null;
    }
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return exitCode;
}

public final java.lang.String switchStatement(AST _t) throws RecognitionException {
    java.lang.String exitCode;

    AST switchStatement_AST_in = (AST)_t;
    AST switchNode = null;

    exitCode = null;

    try {      // for error handling
        switchNode = (AST)_t;
        match(_t,SWITCH);
        _t = _t.getNextSibling();

        AST exprNode = switchNode.getFirstChild();
        AbstractRepresentation as = syntab.getRepresentation(exprNode.toString());
        AbstractRepresentation abRep = null;
        java.lang.String typeName;

        if (as != null)
            typeName = as.getType().getName();
        else
            typeName = "expression";

        java.lang.String valueToMatch;
        if (typeName.equals("char"))
            valueToMatch = "'" + ((CharRepresentation) as).toString() + "'";
        else if (typeName.equals("int"))
            valueToMatch = new java.lang.String(((Integer)((IntRepresentation) as).getValue()).toString());
        else if (typeName.equals("expression")) {
            abRep = expression(exprNode);
            long vTM = ((LongRepresentation) abRep).getValue();
            valueToMatch = new java.lang.String(new Long(vTM).toString());
        } else
            return exitCode;

        debugStream.println(" Attempting to match: " + valueToMatch);
        syntab.beginLevel();

        boolean breakFound = false;
        boolean flowThrough = false;
        AST groupNode = exprNode.getNextSibling(); // these should be CASE_GROUP nodes

```

```

while(groupNode != null && !breakFound) { // iterate over the CASE_GROUP nodes
  AST labelNode = groupNode.getFirstChild();
  while (labelNode != null && !breakFound) {
    if (labelNode.getText().equals("case")) {
      AST labelValue = labelNode.getFirstChild();

      debugStream.println(" Examining case statement: " + labelValue.getText());
      debugStream.println(" Attempting to match: " + valueToMatch + " and " + labelValue.getText());

      // Clean up comparison here, take into account char and int literals
      // (not a problem, as much since we use the string value of the node
      // and obtain the value of a char or int constant identifier.... something we never did....
      String labelType = getType(labelValue.getText());
      String labelToMatch = null;

      // Obtain the label's value, be wary of literals and constant identifiers!
      if (labelType != null) {
        if (labelType.equals("int")) {
          // Handle case of constant int identifier
          IntRepresentation intRep = ((IntRepresentation)
            symtab.getRepresentation(labelValue.getText()));
          labelToMatch = (new Integer(intRep.getValue())).toString();
        }
        else if (labelType.equals("char")) {
          // Handle case of constant character identifier
          CharRepresentation charRep = ((CharRepresentation)
            symtab.getRepresentation(labelValue.getText()));
          labelToMatch = '\'' + (new Character((char) charRep.getValue())).toString() + '\'';
        }
      }
      else if (labelType == null) {
        // It is a char or int literal, so use the node's value

        // If it's a char literal, the second char is a ',
        if (labelValue.getText().charAt(0) == '\''
          if (labelValue.getText().charAt(1) == '\\')
            switch (labelValue.getText().charAt(2)) {
              case 'n':
                labelToMatch = "\"" + '\ n' + "\"";
                break;

              case 'r':
                labelToMatch = "\"" + '\ r' + "\"";
                break;

              case 't':
                labelToMatch = "\"" + '\ t' + "\"";
                break;
            }
          else
            labelToMatch = labelValue.getText();
        else
          labelToMatch = labelValue.getText();
      }
    }
    if (valueToMatch.equals(labelToMatch) || flowThrough) {
      // Match found, skip until we see the SList and then process the list.
      debugStream.println(" Executing case statement: " + labelValue.toString());
      labelNode = labelNode.getNextSibling();
      while (labelNode != null && !labelNode.getText().equals("SLIST"))
        labelNode = labelNode.getNextSibling();

      if (labelNode != null && labelNode.getText().equals("SLIST")) {
        // this should never be null, there should always be a SLIST to find
        AST SListNode = labelNode.getFirstChild();
        while (SListNode != null && !breakFound) {
          exitCode = statement(SListNode);
          if (exitCode != null && (exitCode.equals("break") || exitCode.equals("return"))) {
            breakFound = true;
            flowThrough = false;
          } else
            flowThrough = true;
          SListNode = SListNode.getNextSibling();
        }
      }
    }
  }
}

```

```

    }
    } else if (labelNode.getText().equals("default")) {
        debugStream.println("  Executing default statement.");
        AST SListNode = labelNode.getNextSibling().getFirstChild();
        // picks up the first statement of the SLIST
        while (SListNode != null) {
            exitCode = statement(SListNode);
            SListNode = SListNode.getNextSibling();
        }
    }
    labelNode = labelNode.getNextSibling();
}
groupNode = groupNode.getNextSibling();
}
syntab.endLevel();

// Reset the exit code unless we saw a return in there... (from actual student implementation)
if (exitCode != null && !exitCode.equals("return"))
    exitCode = null;

}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return exitCode;
}

public final void expressionStatement(AST _t) throws RecognitionException {

    AST expressionStatement_AST_in = (AST)_t;

    // this item is only used to store the value so that no warning is thrown during compile time
    AbstractRepresentation z = null;

    try { // for error handling
        AST __t31 = _t;
        AST tmp6_AST_in = (AST)_t;
        match(_t,EXPRESSION_STATEMENT);
        _t = _t.getFirstChild();
        z=expression(_t);
        _t = _retTree;
        _t = __t31;
        _t = _t.getNextSibling();
    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
}

public final java.lang.String blockStatement(AST _t) throws RecognitionException {
    java.lang.String exitCode;

    AST blockStatement_AST_in = (AST)_t;
    AST blockNode = null;

    exitCode = null;

    try { // for error handling
        blockNode = (AST)_t;
        match(_t,BLOCK_STATEMENT);
        _t = _t.getNextSibling();

        AST stmtNode;
        stmtNode = blockNode.getFirstChild();
        syntab.beginLevel();
        while (stmtNode != null) {
            debugStream.println("Executing statement: " + stmtNode);
            exitCode = statement(stmtNode);

```

```

        if (exitCode != null) break;

        stmtNode = stmtNode.getNextSibling();
    }
    symtab.endLevel();
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return exitCode;
}

public final java.lang.String returnStatement(AST _t) throws RecognitionException {
    java.lang.String exitCode;

    AST returnStatement_AST_in = (AST)_t;

    exitCode = "return";
    AbstractRepresentation expr = null;
    AbstractRepresentation rep = null;

    try {        // for error handling
        AST __t19 = _t;
        AST tmp7_AST_in = (AST)_t;
        match(_t,RETURN);
        _t = _t.getFirstChild();
        {
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType() ) {
            case FLOATING_CONSTANT:
            case FALSE:
            case TRUE:
            case LPAREN:
            case INCR:
            case DECR:
            case STAR:
            case PLUS:
            case MINUS:
            case BANG:
            case SLASH:
            case PERCENT:
            case LSHIFT:
            case RSHIFT:
            case LT:
            case GT:
            case LE:
            case GE:
            case EQ:
            case NE:
            case LAND:
            case LOR:
            case ASSIGN:
            case STAR_ASSIGN:
            case SLASH_ASSIGN:
            case PERCENT_ASSIGN:
            case PLUS_ASSIGN:
            case MINUS_ASSIGN:
            case COMMA:
            case IDENTIFIER:
            case INTEGER_CONSTANT:
            case CHAR_CONSTANT:
            case STRING_LITERAL:
            case CALL:
            case CAST:
            case POST_DECR:
            case POST_INCR:
            {
                expr=expression(_t);
                _t = _retTree;
                break;
            }
        }
    }
}

```

```

    }
    case 3:
    {
        break;
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
    }
    }
    _t = _t19;
    _t = _t.getNextSibling();

    FunctionWrapper currentFunction = (FunctionWrapper) functionStack.pop();
    AbstractType typeToReturn = ((TypeRepresentation)
        symtab.getRepresentation(currentFunction.getReturnType()).getRepresentedType());
    try {
        if(expr != null)
            rep = typeManager.convert(expr, typeToReturn);
    } catch(IncompatibleTypesException ex) {
    }
    symtab.setRepresentation("__return", rep);
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return exitCode;
}

public final java.lang.String breakStatement(AST _t) throws RecognitionException {
    java.lang.String exitCode;

    AST breakStatement_AST_in = (AST)_t;

    exitCode = "break";

    try {        // for error handling
        AST tmp8_AST_in = (AST)_t;
        match(_t,BREAK);
        _t = _t.getNextSibling();
    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
    return exitCode;
}

public final java.lang.String continueStatement(AST _t) throws RecognitionException {
    java.lang.String exitCode;

    AST continueStatement_AST_in = (AST)_t;

    exitCode = "continue";

    try {        // for error handling
        AST tmp9_AST_in = (AST)_t;
        match(_t,CONTINUE);
        _t = _t.getNextSibling();
    }
    catch (RecognitionException ex) {
        reportError(ex);
        if (_t!=null) {_t = _t.getNextSibling();}
    }
    _retTree = _t;
    return exitCode;
}
}

```



```

        IFStreamRepresentation isRep = (IFStreamRepresentation) sRep;
        sRepClass = IFStreamRepresentation.class;
        Method methodToCall = sRepClass.getMethod(funcName, actTypes);
        returnVal = methodToCall.invoke(isRep, actValues);
    }

    if(returnVal != null)
    {
        if(returnVal instanceof Boolean)
            exprRep = new BoolRepresentation( ((Boolean) returnVal).booleanValue() );
        else if(returnVal instanceof Character)
            exprRep = new CharRepresentation( ((Character) returnVal).charValue() );
    }
}
catch(NoSuchMethodException nsme) {
    debugStream.println("  no such method exception.");

    // OK, we didn't find an exact match for the intended function call,
    // but we know a suitable one exists if it got through the semantic checker...
    // now we just need to jump through the hoops again to find it!
    Method[] streamMethods = sRepClass.getMethods();
    Class[] streamParamTypes = null;
    int bestFitIndex = -1;
    int leastConversions = 999;
    for (int index=0; index < streamMethods.length; index++) {
        // Since we get ALL public methods, we need to check both the names and the paramlist size
        if (streamMethods[index].getName().equals(funcName) &&
            streamMethods[index].getParameterTypes().length == actTypes.length) {
            // Start worrying finding each possible matching function here
            streamParamTypes = streamMethods[index].getParameterTypes();
            int sumConversions = 0;
            for (int paramIndex = 0; paramIndex < streamParamTypes.length; paramIndex++)
                sumConversions += typeManager.checkConversionSteps(
                    NewSemantic.convertClassToAbstract(streamParamTypes[paramIndex]),
                    ((AbstractRepresentation) actValues[paramIndex]).getType());

            // Keep a reference to the single best match found
            if (sumConversions > 0 && sumConversions < leastConversions) {
                leastConversions = sumConversions;
                bestFitIndex = index;
            }
        }
    }
    if (bestFitIndex != -1) {
        // If we've found the match, run with it!
        Method streamMethod = streamMethods[bestFitIndex];
        try {
            System.out.println("found it.");
            streamMethod.invoke(null, getConvertedActualValues((SandboxAST)
                streamNode.getNextSibling(), streamParamTypes));
        } catch (Exception e) {
            debugStream.println("...exception caught during invocation of function.");
            System.err.println("Exception caught during invocation of function.");
            e.printStackTrace();
        }
    }
}
catch(InvocationTargetException e) {
    debugStream.println("...exception caught during invocation of function.");
    e.printStackTrace();
}
catch(Exception e) {
    // This IS something to worry about; shouldn't get here.
    e.printStackTrace();
}
} else if (typeName != null && typeName.equals("string")) {
    // Handle method calls on string objects.

    // Reads and ignores the 'dot' in the tree
    tempNode = (SandboxAST) streamNode.getNextSibling();

    // Reads the name of the method call
    callName = (SandboxAST) tempNode.getNextSibling();
}

```

```

// Reads the arguments list
SandboxAST argsList = (SandboxAST) callName.getNextSibling();

// Attempt to find the function in a library and then invoke it if found.
try {
    StringRepresentation stringRepresentation = (StringRepresentation)
        symtab.getRepresentation(streamNode.getText());
    Method methodToCall = StringRepresentation.class.getDeclaredMethod(
        callName.getText(), new Class[] { });
    Object valueReturned = methodToCall.invoke(stringRepresentation, new Object[] { });
    exprRep = new IntRepresentation(((Integer) valueReturned).intValue());
}
catch(NoSuchMethodException e) {
    // This would be bad...
    e.printStackTrace( System.out );
}
catch(Exception e) {
    // This IS something to worry about; shouldn't get here.
    e.printStackTrace( System.out );
}
} else if (typeName != null && (typeName.equals("istream"))) {
    // Handle method calls on istream objects.

    // Reads and ignores the 'dot' in the tree
    tempNode = (SandboxAST) streamNode.getNextSibling();

    // Reads the name of the method call
    callName = (SandboxAST) tempNode.getNextSibling();

    // Reads the arguments list
    SandboxAST argsList = (SandboxAST) callName.getNextSibling();

    // retrieve the stream to use.
    IStreamRepresentation streamRepresentation = (IStreamRepresentation)
        symtab.getRepresentation(streamNode.getText());

    try
    {
        Class sRepClass = IStreamRepresentation.class;
        String funcName = callName.getText();
        Class[] actTypes = getActualTypes((SandboxAST) argsList);
        Object[] actValues = getActualValues((SandboxAST) argsList);
        try
        {
            Method methodToCall = sRepClass.getMethod(funcName, actTypes);
            methodToCall.invoke(streamRepresentation, actValues);
        }
        catch( Exception e )
        {
            e.printStackTrace( System.out );
        }
    } catch (Exception io) {
        debugStream.println("..exception caught");
        System.err.println("IStream failed");
        io.printStackTrace();
    }
}

// Handle calls to user-defined functions first (if possible), and then libraries.
} else {
    String funcName = streamNode.getText();
    tempNode = (SandboxAST) streamNode.getNextSibling(); // gets the "ARGUMENTS" node
    AbstractRepresentation[] repArray = null;

    FunctionWrapper function = null;
    if (tempNode != null) {
        repArray = arguments(tempNode);
        function = fm.locateFunction(funcName, repArray.length);
    } else
        function = fm.locateFunction(funcName, 0);

    if (function != null) {
        if (function.getFunctionRoot() != null) {
            // The call is to a user-defined function
            debugStream.println("Calling user-defined function: " + funcName);

```

```

    exprRep = function.invoke(this, repArray);
  } else {
    reportError(funcName, ErrorCodeTable.UNABLE_TO_LOCATE_FUNCTION,
      ((SandboxAST) streamNode).getLine(), 0);
    debugStream.println(funcName + " function not found, possible scope issue...");
    // Don't reset exprRep so that a null is returned. This is akin to a linker error,
    // but we don't want anything
    // meaningful to happen... may result in an internal exception being thrown,
    // but that's ok...
  }
} else if (libraryManager.containsFunction(funcName)) {
  // The call is to a library function
  debugStream.println("Calling library function: " + funcName);

  // Ensure what's in the library is call-able, otherwise, it's likely to be
  // handled here in the interpreter, above, as a special function (for I/O)!
  if (libraryManager.inLibrary(funcName)) {
    debugStream.println("  " + funcName + " found in library file.");
    Class libClass = null;
    Class[] actTypes = getActualTypes((SandboxAST) streamNode.getNextSibling());

    try {
      libClass = Class.forName(libraryManager.getOwner(funcName));
      Method libMethod = libClass.getMethod(funcName, actTypes);
      SandboxAST paramStartNode = (SandboxAST) streamNode.getNextSibling();
      Object args[] = getActualValues(paramStartNode);
      Object valueReturned = libMethod.invoke(null, args);

      String returnType = libMethod.getReturnType().getName();
      // add CHARS here and test!
      if (returnType.equals("double")) {
        exprRep = new DoubleRepresentation(((Double) valueReturned).doubleValue());
        if (Double.isNaN(((DoubleRepresentation) exprRep).getValue())) {
          reportError(null, ErrorCodeTable.NOT_A_NUMBER, ((SandboxAST)
            callNode.getFirstChild()).getLine(), 0);
          debugStream.println("  Run time error: expression value is not a number.");
        }
      }

      } else if (returnType.equals("int"))
        exprRep = new IntRepresentation(((Integer) valueReturned).intValue());

      else if (returnType.equals("boolean"))
        exprRep = new BoolRepresentation(((Boolean) valueReturned).booleanValue());

      else if (returnType.equals("void"))
        exprRep = null;

      if (libMethod.getName().equals("getline")) {
        // Replace the string destination (2nd formal parameter) with the returned value
        SandboxAST paramNode = (SandboxAST)
          streamNode.getNextSibling().getFirstChild().getNextSibling();
        StringRepresentation rep = (StringRepresentation)
          symtab.getRepresentation(paramNode.getText());
        rep.setValue(((StringRepresentation) args[1]).getValue());
        exprRep = null; // Force function to return a null, since we overrode
        // it with String to get modified value out...
      }
    } catch (NoSuchMethodException nsme) {
      debugStream.println("  no such method exception.");

      // OK, we didn't find an exact match for the intended function call,
      // but we know a suitable one exists if it got through the semantic checker...
      // now we just need to jump through the hoops again to find it!
      Method[] libMethods = libClass.getMethods();
      Class[] libParamTypes = null;
      int bestFitIndex = -1;
      int leastConversions = 999;
      for (int index=0; index < libMethods.length; index++) {
        // Since we get ALL public methods, we need to check both the names and the paramlist size
        if (libMethods[index].getName().equals(funcName) &&
          libMethods[index].getParameterTypes().length == actTypes.length) {
          // Start worrying finding each possible matching function here
          libParamTypes = libMethods[index].getParameterTypes();
        }
      }
    }
  }
}

```

```

        int sumConversions = 0;
        for (int paramIndex = 0; paramIndex < libParamTypes.length; paramIndex++)
            sumConversions += typeManager.checkConversionSteps(
                NewSemantic.convertClassToAbstract(libParamTypes[paramIndex]),
                NewSemantic.convertClassToAbstract(actTypes[paramIndex]));

        // Keep a reference to the single best match found
        if (sumConversions > 0 && sumConversions < leastConversions) {
            leastConversions = sumConversions;
            bestFitIndex = index;
        }
    }
}
if (bestFitIndex != -1) {
    // If we've found the match, run with it!
    Method libMethod = libMethods[bestFitIndex];
    try {
        Object valueReturned = libMethod.invoke(null,
            getConvertedActualValues((SandboxAST)
                streamNode.getNextSibling(), libParamTypes));

        String returnType = libMethod.getReturnType().getName();
        // add CHARS here and test!

        // Convert what we get back (java wrapper type) to a C++ type...
        if (returnType.equals("double")) {
            exprRep = new DoubleRepresentation(((Double) valueReturned).doubleValue());
            if (Double.isNaN(((DoubleRepresentation) exprRep).getValue())) {
                reportError(null, ErrorCodeTable.NOT_A_NUMBER,
                    ((SandboxAST) callNode.getFirstChild()).getLine(), 0);
                debugStream.println(" Run time error: expression value is not a number.");
            }
        }
        else if (returnType.equals("int"))
            exprRep = new IntRepresentation(((Integer) valueReturned).intValue());
        else if (returnType.equals("boolean"))
            exprRep = new BoolRepresentation(((Boolean) valueReturned).booleanValue());
    } catch (Exception e) {
        debugStream.println("...exception caught during invocation of function.");
        System.err.println("Exception caught during invocation of function.");
        e.printStackTrace();
    }
}
} catch (Exception e) {
    debugStream.println("...exception caught");
    System.err.println("Exception while obtaining library function:" + funcName);
    e.printStackTrace();
}
} else
    // Return a string representation with the called function's name in it.
    // This is how we handle those special case functions, which eventually will need
    // to be migrated over.
    exprRep = new StringRepresentation(funcName);
}
}

break;
}
case CAST:
{
    AST __t41 = _t;
    AST tmp10_AST_in = (AST)_t;
    match(_t,CAST);
    _t = _t.getFirstChild();
    typeNode = (AST)_t;
    match(_t,TYPENAME);
    _t = _t.getNextSibling();
    a=expression(_t);
    _t = _retTree;
    _t = __t41;
    _t = _t.getNextSibling();

    AbstractRepresentation typeRep = syntab.getRepresentation(typeNode.getText());

    if (typeRep != null && typeRep instanceof TypeRepresentation) {

```

```

        AbstractType type = ((TypeRepresentation) typeRep).getRepresentedType();

        try {
            exprRep = typeManager.convert(a, type);
        } catch(IncompatibleTypesException ex) {
        }
    }

    break;
}
case COMMA:
{
    AST _t42 = _t;
    commaNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,COMMA);
    _t = _t.getFirstChild();
    {
        a=expression(_t);
        _t = _retTree;
        b=expression(_t);
        _t = _retTree;
    }
    _t = _t42;
    _t = _t.getNextSibling();
    break;
}
case ASSIGN:
{
    AST _t44 = _t;
    assignNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,ASSIGN);
    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    b=expression(_t);
    _t = _retTree;
    _t = _t44;
    _t = _t.getNextSibling();

    AST leftNode = assignNode.getFirstChild();
    RepresentationRecord rr = symtab.getRepresentationRecord(leftNode.getText());

    if (rr != null && rr.getRepresentation().isConstant()) {
        reportError(leftNode.getText() + " is a constant; it can not be modified.",
            ErrorCodeTable.RUNTIME_ASSIGNMENT_TO_CONSTANT, ((SandboxAST) leftNode).getLine(), 0);
        debugStream.println("    Run time error: attempting to modify a constant.");
    } else
    {
        exprRep = evaluateAssignment(ASSIGN, a, b);
    }

    break;
}
case PLUS_ASSIGN:
{
    AST _t45 = _t;
    pAssignNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,PLUS_ASSIGN);
    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    b=expression(_t);
    _t = _retTree;
    _t = _t45;
    _t = _t.getNextSibling();

    b = evaluateBinaryOp(PLUS, a, b);
    AST leftNode = pAssignNode.getFirstChild();
    RepresentationRecord rr = symtab.getRepresentationRecord(leftNode.getText());

    if (rr != null && rr.getRepresentation().isConstant()) {
        reportError(leftNode.getText() + " is a constant; it can not be modified.",
            ErrorCodeTable.RUNTIME_ASSIGNMENT_TO_CONSTANT, ((SandboxAST) leftNode).getLine(), 0);
        debugStream.println("    Run time error: attempting to modify a constant.");
    }
}

```

```

        } else
            exprRep = evaluateAssignment(PLUS_ASSIGN, a, b);

        break;
    }
    case MINUS_ASSIGN:
    {
        AST __t46 = _t;
        mAssignNode = _t==ASTNULL ? null :(AST)_t;
        match(_t,MINUS_ASSIGN);
        _t = _t.getFirstChild();
        a=expression(_t);
        _t = _retTree;
        b=expression(_t);
        _t = _retTree;
        _t = __t46;
        _t = _t.getNextSibling();

        b = evaluateBinaryOp(MINUS, a, b);
        AST leftNode = mAssignNode.getFirstChild();
        RepresentationRecord rr = symtab.getRepresentationRecord(leftNode.getText());

        if (rr != null && rr.getRepresentation().isConstant()) {
            reportError(leftNode.getText() + " is a constant; it can not be modified.",
                ErrorCodeTable.RUNTIME_ASSIGNMENT_TO_CONSTANT, ((SandboxAST) leftNode).getLine(), 0);
            debugStream.println("    Run time error: attempting to modify a constant.");
        } else
            exprRep = evaluateAssignment(MINUS_ASSIGN, a, b);

        break;
    }
    case STAR_ASSIGN:
    {
        AST __t47 = _t;
        sAssignNode = _t==ASTNULL ? null :(AST)_t;
        match(_t,STAR_ASSIGN);
        _t = _t.getFirstChild();
        a=expression(_t);
        _t = _retTree;
        b=expression(_t);
        _t = _retTree;
        _t = __t47;
        _t = _t.getNextSibling();

        b = evaluateBinaryOp(STAR, a, b);
        AST leftNode = sAssignNode.getFirstChild();
        RepresentationRecord rr = symtab.getRepresentationRecord(leftNode.getText());

        if (rr != null && rr.getRepresentation().isConstant()) {
            reportError(leftNode.getText() + " is a constant; it can not be modified.",
                ErrorCodeTable.RUNTIME_ASSIGNMENT_TO_CONSTANT, ((SandboxAST) leftNode).getLine(), 0);
            debugStream.println("    Run time error: attempting to modify a constant.");
        } else
            exprRep = evaluateAssignment(STAR_ASSIGN, a, b);

        break;
    }
    case SLASH_ASSIGN:
    {
        AST __t48 = _t;
        slAssignNode = _t==ASTNULL ? null :(AST)_t;
        match(_t,SLASH_ASSIGN);
        _t = _t.getFirstChild();
        a=expression(_t);
        _t = _retTree;
        b=expression(_t);
        _t = _retTree;
        _t = __t48;
        _t = _t.getNextSibling();

        b = evaluateBinaryOp(SLASH, a, b);
        AST leftNode = slAssignNode.getFirstChild();
        RepresentationRecord rr = symtab.getRepresentationRecord(leftNode.getText());

```

```

        if (rr != null && rr.getRepresentation().isConstant()) {
            reportError(leftNode.getText() + " is a constant; it can not be modified.",
                ErrorCodeTable.RUNTIME_ASSIGNMENT_TO_CONSTANT, ((SandboxAST) leftNode).getLine(), 0);
            debugStream.println("    Run time error: attempting to modify a constant.");
        } else
            exprRep = evaluateAssignment(SLASH_ASSIGN, a, b);

        break;
    }
    case PERCENT_ASSIGN:
    {
        AST __t49 = _t;
        perAssignNode = _t==ASTNULL ? null :(AST)_t;
        match(_t,PERCENT_ASSIGN);
        _t = _t.getFirstChild();
        a=expression(_t);
        _t = _retTree;
        b=expression(_t);
        _t = _retTree;
        _t = __t49;
        _t = _t.getNextSibling();

        b = evaluateBinaryOp(PERCENT, a, b);
        AST leftNode = perAssignNode.getFirstChild();
        RepresentationRecord rr = sytab.getRepresentationRecord(leftNode.getText());

        if (rr != null && rr.getRepresentation().isConstant()) {
            reportError(leftNode.getText() + " is a constant; it can not be modified.",
                ErrorCodeTable.RUNTIME_ASSIGNMENT_TO_CONSTANT, ((SandboxAST) leftNode).getLine(), 0);
            debugStream.println("    Run time error: attempting to modify a constant.");
        } else
            exprRep = evaluateAssignment(PERCENT_ASSIGN, a, b);

        break;
    }
    case EQ:
    {
        AST __t50 = _t;
        AST tmp11_AST_in = (AST)_t;
        match(_t,EQ);
        _t = _t.getFirstChild();
        a=expression(_t);
        _t = _retTree;
        b=expression(_t);
        _t = _retTree;
        _t = __t50;
        _t = _t.getNextSibling();

        // Check for stream types
        if ((a != null) && (b != null)) {
            // The stream MUST be compared to a bool. Conversion is not necessary
            if (a.getType().isStream() && (b instanceof BoolRepresentation)) {
                boolean isFailed = ((Stream)a).getFailed();
                boolean boolValue= ((BoolRepresentation)b).getValue();
                exprRep= new BoolRepresentation(!isFailed == boolValue);
            }
            else if (b.getType().isStream() && (a instanceof BoolRepresentation)) {
                boolean isFailed = ((Stream)b).getFailed();
                boolean boolValue= ((BoolRepresentation)a).getValue();
                exprRep= new BoolRepresentation(!isFailed == boolValue);
            }
            else
                exprRep = evaluateBinaryOp(EQ, a, b);
        }
        else
            exprRep = evaluateBinaryOp(EQ, a, b);

        break;
    }
    case NE:
    {
        AST __t51 = _t;
        AST tmp12_AST_in = (AST)_t;
        match(_t,NE);

```

```

    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    b=expression(_t);
    _t = _retTree;
    _t = _t51;
    _t = _t.getNextSibling();

    // Check for stream types
    if ((a != null) && (b != null)) {
        // The stream MUST be compared to a bool. Conversion is not necessary
        if (a.getType().isStream() && (b instanceof BoolRepresentation)) {
            boolean isFailed = ((Stream)a).getFailed();
            boolean boolValue= ((BoolRepresentation)b).getValue();
            exprRep= new BoolRepresentation(!isFailed != boolValue);
        }
        else if (b.getType().isStream() && (a instanceof BoolRepresentation)) {
            boolean isFailed = ((Stream)b).getFailed();
            boolean boolValue= ((BoolRepresentation)a).getValue();
            exprRep= new BoolRepresentation(!isFailed != boolValue);
        }
        else
            exprRep = evaluateBinaryOp(NE, a, b);
    }
    else
        exprRep = evaluateBinaryOp(NE, a, b);

    break;
}
case LT:
{
    AST _t52 = _t;
    AST tmp13_AST_in = (AST)_t;
    match(_t,LT);
    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    b=expression(_t);
    _t = _retTree;
    _t = _t52;
    _t = _t.getNextSibling();

    exprRep = evaluateBinaryOp(LT, a, b);

    break;
}
case GT:
{
    AST _t53 = _t;
    AST tmp14_AST_in = (AST)_t;
    match(_t,GT);
    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    b=expression(_t);
    _t = _retTree;
    _t = _t53;
    _t = _t.getNextSibling();

    exprRep = evaluateBinaryOp(GT, a, b);

    break;
}
case LE:
{
    AST _t54 = _t;
    AST tmp15_AST_in = (AST)_t;
    match(_t,LE);
    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    b=expression(_t);
    _t = _retTree;
    _t = _t54;

```

```

    _t = _t.getNextSibling();

    exprRep = evaluateBinaryOp(LE, a, b);

    break;
}
case GE:
{
    AST __t55 = _t;
    AST tmp16_AST_in = (AST)_t;
    match(_t,GE);
    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    b=expression(_t);
    _t = _retTree;
    _t = __t55;
    _t = _t.getNextSibling();

    exprRep = evaluateBinaryOp(GE, a, b);

    break;
}
case POST_INCR:
{
    AST __t56 = _t;
    AST tmp17_AST_in = (AST)_t;
    match(_t,POST_INCR);
    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    _t = __t56;
    _t = _t.getNextSibling();

    b = evaluateBinaryOp(PLUS, a, ONE);
    exprRep = a.cloneRepresentation();
    evaluateAssignment(ASSIGN, a, b);

    break;
}
case POST_DECR:
{
    AST __t57 = _t;
    AST tmp18_AST_in = (AST)_t;
    match(_t,POST_DECR);
    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    _t = __t57;
    _t = _t.getNextSibling();

    b = evaluateBinaryOp(MINUS, a, ONE);
    exprRep = a.cloneRepresentation();
    evaluateAssignment(ASSIGN, a, b);

    break;
}
case INCR:
{
    AST __t58 = _t;
    AST tmp19_AST_in = (AST)_t;
    match(_t,INCR);
    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    _t = __t58;
    _t = _t.getNextSibling();

    b = evaluateBinaryOp(PLUS, a, ONE);
    exprRep = evaluateAssignment(ASSIGN, a, b);
    exprRep = a;

    break;
}

```

```

case DECR:
{
  AST __t59 = _t;
  AST tmp20_AST_in = (AST)_t;
  match(_t,DECR);
  _t = _t.getFirstChild();
  a=expression(_t);
  _t = _retTree;
  _t = __t59;
  _t = _t.getNextSibling();

      b = evaluateBinaryOp(MINUS, a, ONE);
      exprRep = evaluateAssignment(ASSIGN, a, b);
      exprRep = a;

  break;
}
case PLUS:
{
  AST __t60 = _t;
  AST tmp21_AST_in = (AST)_t;
  match(_t,PLUS);
  _t = _t.getFirstChild();
  a=expression(_t);
  _t = _retTree;
  b=expression(_t);
  _t = _retTree;
  _t = __t60;
  _t = _t.getNextSibling();

      exprRep = evaluateBinaryOp(PLUS, a, b);

  break;
}
case MINUS:
{
  AST __t61 = _t;
  AST tmp22_AST_in = (AST)_t;
  match(_t,MINUS);
  _t = _t.getFirstChild();
  a=expression(_t);
  _t = _retTree;
  {
    if (_t==null) _t=ASTNULL;
    switch ( _t.getType() ) {
    case FLOATING_CONSTANT:
    case FALSE:
    case TRUE:
    case LPAREN:
    case INCR:
    case DECR:
    case STAR:
    case PLUS:
    case MINUS:
    case BANG:
    case SLASH:
    case PERCENT:
    case LSHIFT:
    case RSHIFT:
    case LT:
    case GT:
    case LE:
    case GE:
    case EQ:
    case NE:
    case LAND:
    case LOR:
    case ASSIGN:
    case STAR_ASSIGN:
    case SLASH_ASSIGN:
    case PERCENT_ASSIGN:
    case PLUS_ASSIGN:
    case MINUS_ASSIGN:
    case COMMA:

```

```

    case IDENTIFIER:
    case INTEGER_CONSTANT:
    case CHAR_CONSTANT:
    case STRING_LITERAL:
    case CALL:
    case CAST:
    case POST_DECR:
    case POST_INCR:
    {
        b=expression(_t);
        _t = _retTree;
        break;
    }
    case 3:
    {
        break;
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
    }
    _t = _t61;
    _t = _t.getNextSibling();

        exprRep = evaluateBinaryOp(MINUS, a, b);

    break;
}
case STAR:
{
    AST _t63 = _t;
    AST tmp23_AST_in = (AST)_t;
    match(_t,STAR);
    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    b=expression(_t);
    _t = _retTree;
    _t = _t63;
    _t = _t.getNextSibling();

        exprRep = evaluateBinaryOp(STAR, a, b);

    break;
}
case SLASH:
{
    AST _t64 = _t;
    s1 = _t==ASTNULL ? null :(AST)_t;
    match(_t,SLASH);
    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    b=expression(_t);
    _t = _retTree;
    _t = _t64;
    _t = _t.getNextSibling();

        currentLineNumber = ((SandboxAST) s1).getLine();
        exprRep = evaluateBinaryOp(SLASH, a, b);

    break;
}
case PERCENT:
{
    AST _t65 = _t;
    pct = _t==ASTNULL ? null :(AST)_t;
    match(_t,PERCENT);
    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    b=expression(_t);

```

```

    _t = _retTree;
    _t = _t65;
    _t = _t.getNextSibling();

    currentLineNumber = ((SandboxAST) pct).getLine();
    exprRep = evaluateBinaryOp(PERCENT, a, b);

    break;
}
case LSHIFT:
{
    AST _t66 = _t;
    lshiftNode = _t==ASTNULL ? null :(AST)_t;
    match(_t,LSHIFT);
    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    b=expression(_t);
    _t = _retTree;
    _t = _t66;
    _t = _t.getNextSibling();

    // Handling the case of calling a stream method...
    if (a != null && (a.getType().getName().equals("ofstream") || (a.getType().getName().equals("ostream")))
        && b != null && b.getType().getName().equals("string")) {

        StringRepresentation bstrRep = (StringRepresentation) b;
        // Handle calling a setprecision on a file or output stream

        if (bstrRep.getValue() != null)
        {
            String manipName = bstrRep.getValue();
            OStreamRepresentation osRep = (OStreamRepresentation) a;

            AST callNode2 = lshiftNode.getFirstChild(); // gets the ofstream node
            callNode2 = callNode2.getNextSibling(); // gets the next node (CALL or identifier)

            if (callNode2.getText().equals("CALL"))
            {
                callNode2 = callNode2.getFirstChild(); // gets the manip node
                callNode2 = callNode2.getNextSibling(); // gets the ARGUMENTS and parameter nodes

                Class[] actTypes = getActualTypes( (SandboxAST) callNode2 );
                Object[] actVals = getActualValues( (SandboxAST) callNode2 );

                try {
                    Method methodToCall = OStreamRepresentation.class.getMethod(manipName, actTypes);
                    methodToCall.invoke(osRep, actVals);
                    exprRep = osRep;
                }
                catch(NoSuchMethodException e) {
                    // No problem; wasn't a setprecision, setw, or setfill call. Just carry on.
                    exprRep = evaluateBinaryOp(LSHIFT, a, b);
                }
                catch(Exception e) { /* This won't happen. */ }
            }
            else
                exprRep = evaluateBinaryOp(LSHIFT, a, b);
        }
    }
    else
        exprRep = evaluateBinaryOp(LSHIFT, a, b);

    break;
}
case RSHIFT:
{
    AST _t67 = _t;
    AST tmp24_AST_in = (AST)_t;
    match(_t,RSHIFT);
    _t = _t.getFirstChild();
    a=expression(_t);
    _t = _retTree;
    b=expression(_t);

```

```

    _t = _retTree;
    _t = _t67;
    _t = _t.getNextSibling();

    exprRep = evaluateBinaryOp(RSHIFT, a, b);
    if (a.getType().getName().equals("istream") || a.getType().getName().equals("ifstream"))
        // Return the "correct" result, though we converted to another data type...
        // this permits following expressions (if any) to utilize i*streams in the expression
        exprRep = a;

    break;
}
case FLOATING_CONSTANT:
case FALSE:
case TRUE:
case LPAREN:
case IDENTIFIER:
case INTEGER_CONSTANT:
case CHAR_CONSTANT:
case STRING_LITERAL:
{
    exprRep=primaryExpr(_t);
    _t = _retTree;
    break;
}
case BANG:
{
    bangNode = (AST)_t;
    match(_t,BANG);
    _t = _t.getNextSibling();

    boolean result = evaluateBoolExpression(bangNode.getFirstChild());

    debugStream.println("  BANG operator, turning " + result + " into " + !result);
    exprRep = new BoolRepresentation(!result);

    break;
}
case LAND:
{
    landNode = (AST)_t;
    match(_t,LAND);
    _t = _t.getNextSibling();

    debugStream.println("  evaluating &&");

    AST node = landNode.getFirstChild();
    boolean value = evaluateBoolExpression(node);
    if (value)
        value = evaluateBoolExpression(node.getNextSibling());
    else
        debugStream.println("  evaluation short-circuited");
    exprRep = new BoolRepresentation(value);

    break;
}
case LOR:
{
    lorNode = (AST)_t;
    match(_t,LOR);
    _t = _t.getNextSibling();

    debugStream.println("evaluating ||");

    AST node = lorNode.getFirstChild();
    boolean value = evaluateBoolExpression(node);

    if (!value)
        value = evaluateBoolExpression(node.getNextSibling());
    else
        debugStream.println("  evaluation short-circuited");
    exprRep = new BoolRepresentation(value);

    break;
}

```

```

    }
    default:
    {
        throw new NoViableAltException(_t);
    }
}
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return exprRep;
}

public final void declarator(AST _t,
    java.lang.String typeName, boolean isAConst
) throws RecognitionException {

    AST declarator_AST_in = (AST)_t;
    AST id = null;

    AbstractRepresentation s = null;

    try {        // for error handling
        AST __t34 = _t;
        AST tmp25_AST_in = (AST)_t;
        match(_t,DECLARATOR);
        _t = _t.getFirstChild();
        id = (AST)_t;
        match(_t,IDENTIFIER);
        _t = _t.getNextSibling();
        {
            if (_t==null) _t=ASTNULL;
            switch ( _t.getType() ) {
            case FLOATING_CONSTANT:
            case FALSE:
            case TRUE:
            case LPAREN:
            case INCR:
            case DECR:
            case STAR:
            case PLUS:
            case MINUS:
            case BANG:
            case SLASH:
            case PERCENT:
            case LSHIFT:
            case RSHIFT:
            case LT:
            case GT:
            case LE:
            case GE:
            case EQ:
            case NE:
            case LAND:
            case LOR:
            case ASSIGN:
            case STAR_ASSIGN:
            case SLASH_ASSIGN:
            case PERCENT_ASSIGN:
            case PLUS_ASSIGN:
            case MINUS_ASSIGN:
            case COMMA:
            case IDENTIFIER:
            case INTEGER_CONSTANT:
            case CHAR_CONSTANT:
            case STRING_LITERAL:
            case CALL:
            case CAST:
            case POST_DECR:
            case POST_INCR:
            {

```

```

        s=expression(_t);
        _t = _retTree;
        break;
    }
    case 3:
    {
        break;
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
    }
    }
    _t = _t34;
    _t = _t.getNextSibling();

    AbstractRepresentation typeRep = symtab.getRepresentation(typeName);
    AbstractRepresentation rep = null;
    RepresentationRecord rr = new RepresentationRecord();
    if (isAConst)
        rr.setConstant(true);

    symtab.setRepresentationRecord(id.getText(), rr);

    if (typeRep == null || !(typeRep instanceof TypeRepresentation))
        debugStream.println("    Unknown type: " + typeName);
    else {
        AbstractType type = ((TypeRepresentation) typeRep).getRepresentedType();

        if (s == null)
            rep = type.instantiate();
        else
            try {
                debugStream.println("    Initializing " + id.getText() + " to: " + s.toString());
                rep = typeManager.convert(s, type);
            } catch (IncompatibleTypesException ex) {
                debugStream.println("    Could not cast initializer!");
            }
    }
    symtab.setRepresentation(id.toString(), rep);
    debugStream.println("    Declared " + id.getText() + " as " + rep);

}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
}

public final AbstractRepresentation primaryExpr(AST _t) throws RecognitionException {
    AbstractRepresentation retval;

    AST primaryExpr_AST_in = (AST)_t;
    AST idval = null;

    debugStream.println("    evaluating primaryExpr: " + _t.toStringList());
    retval = null;

    try {        // for error handling
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType()) {
        case IDENTIFIER:
        {
            idval = (AST)_t;
            match(_t, IDENTIFIER);
            _t = _t.getNextSibling();

            retval = symtab.getRepresentation(idval.getText());

            break;
        }
    }
}

```

```

    case FLOATING_CONSTANT:
    case FALSE:
    case TRUE:
    case INTEGER_CONSTANT:
    case CHAR_CONSTANT:
    case STRING_LITERAL:
    {
        retval=constant(_t);
        _t = _retTree;
        break;
    }
    case LPAREN:
    {
        AST __t37 = _t;
        AST tmp26_AST_in = (AST)_t;
        match(_t,LPAREN);
        _t = _t.getFirstChild();
        retval=expression(_t);
        _t = _retTree;
        _t = __t37;
        _t = _t.getNextSibling();
        break;
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
}
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return retval;
}

public final AbstractRepresentation constant(AST _t) throws RecognitionException {
    AbstractRepresentation retval;

    AST constant_AST_in = (AST)_t;
    AST intval = null;
    AST floatval = null;
    AST charval = null;
    AST stringval = null;

    retval = null;

    try { // for error handling
        if (_t==null) _t=ASTNULL;
        switch ( _t.getType()) {
        case INTEGER_CONSTANT:
        {
            intval = (AST)_t;
            match(_t,INTEGER_CONSTANT);
            _t = _t.getNextSibling();

            retval = new LongRepresentation(intval.getText());

            break;
        }
        case FLOATING_CONSTANT:
        {
            floatval = (AST)_t;
            match(_t,FLOATING_CONSTANT);
            _t = _t.getNextSibling();

            retval = new DoubleRepresentation(floatval.getText());

            break;
        }
        case CHAR_CONSTANT:
        {

```

```

        charval = (AST)_t;
        match(_t,CHAR_CONSTANT);
        _t = _t.getNextSibling();

        retval = new CharRepresentation(EscapeTranslator.translateEscapeSequence(charval.getText()).charAt(1));

        break;
    }
    case STRING_LITERAL:
    {
        stringval = (AST)_t;
        match(_t,STRING_LITERAL);
        _t = _t.getNextSibling();

        java.lang.String str = EscapeTranslator.translateEscapeSequence(stringval.getText());
        retval = new StringLiteralRepresentation(str.substring(1, str.length() - 1));

        break;
    }
    case TRUE:
    {
        AST tmp27_AST_in = (AST)_t;
        match(_t,TRUE);
        _t = _t.getNextSibling();

        retval = new BoolRepresentation(true);

        break;
    }
    case FALSE:
    {
        AST tmp28_AST_in = (AST)_t;
        match(_t,FALSE);
        _t = _t.getNextSibling();

        retval = new BoolRepresentation(false);

        break;
    }
    default:
    {
        throw new NoViableAltException(_t);
    }
}
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return retval;
}

public final AbstractRepresentation[] arguments(AST _t) throws RecognitionException {
    AbstractRepresentation[] repArray;

    AST arguments_AST_in = (AST)_t;
    AST argNode = null;

    repArray = null;

    try {        // for error handling
        argNode = (AST)_t;
        match(_t,ARGUMENTS);
        _t = _t.getNextSibling();

        Vector repVector = new Vector();
        AST singleArg = argNode.getFirstChild();
        AbstractRepresentation rep;

        while(singleArg != null) {
            rep = expression(singleArg);

```

```

// Set the AbstractRepresentation to be a constant if the argument is a constant.
// This is a semi-duplication of the constant flag, but since we can't get access
// to the argument's name at the time of the call, we need to pass the info through
// the AbstractRepresentation. Used to check functions attempting to modify a constant
// when passed by reference.
RepresentationRecord rr = symtab.getRepresentationRecord(singleArg.getText());
if (rr != null && rr.isConstant())
    rep.setConstant();

repVector.addElement(rep);
singleArg = singleArg.getNextSibling();
}

int count = repVector.size();
repArray = new AbstractRepresentation[count];
for(int i = 0; i < count; i++)
    repArray[i] = (AbstractRepresentation) repVector.elementAt(i);
}
catch (RecognitionException ex) {
    reportError(ex);
    if (_t!=null) {_t = _t.getNextSibling();}
}
_retTree = _t;
return repArray;
}

public static final String[] _tokenNames = {
    "<0>",
    "EOF",
    "<2>",
    "NULL_TREE_LOOKAHEAD",
    "DOT",
    "FLOATING_CONSTANT",
    "break",
    "case",
    "const",
    "const_cast",
    "continue",
    "default",
    "do",
    "else",
    "enum",
    "for",
    "false",
    "if",
    "namespace",
    "using",
    "return",
    "reinterpret_cast",
    "sizeof",
    "static_cast",
    "struct",
    "switch",
    "TYPENAME",
    "typedef",
    "true",
    "while",
    "LBRACKET",
    "RBRACKET",
    "LPAREN",
    "RPAREN",
    "LBRACE",
    "RBRACE",
    "INCR",
    "DECR",
    "AMPERSAND",
    "STAR",
    "PLUS",
    "MINUS",
    "TILDE",
    "BANG",
    "SLASH",

```

```
"PERCENT",
"LSHIFT",
"RSHIFT",
"LT",
"GT",
"LE",
"GE",
"EQ",
"NE",
"CARET",
"PIPE",
"LAND",
"LOR",
"QUESTION",
"COLON",
"COLON_COLON",
"SEMI",
"ASSIGN",
"STAR_ASSIGN",
"SLASH_ASSIGN",
"PERCENT_ASSIGN",
"PLUS_ASSIGN",
"MINUS_ASSIGN",
"COMMA",
"POUND",
"LSHIFT_ASSIGN",
"RSHIFT_ASSIGN",
"AMPERSAND_ASSIGN",
"CARET_ASSIGN",
"PIPE_ASSIGN",
"WS",
"WHITESPACE",
"NEW_LINE",
"SL_COMMENT",
"ML_COMMENT",
"IDENTIFIER",
"IDENTIFIER_NONDIGIT",
"DIGIT",
"INTEGER_CONSTANT",
"EXPONENT",
"FLOATING_SUFFIX",
"INTEGER_SUFFIX",
"CHAR_CONSTANT",
"STRING_LITERAL",
"ESC",
"HEADER_NAME",
"PP_TOKENS",
"PREPROCESSING_TOKEN",
"POUND_INCLUDE_DIRECTIVE",
"ARGUMENTS",
"ARRAY_DECLARATOR",
"BLOCK_STATEMENT",
"CALL",
"CASE_GROUP",
"CAST",
"DECLARATION",
"DECLARATOR",
"ENUMERATOR_LIST",
"EXPRESSION_STATEMENT",
"FOR_INIT",
"FOR_COND",
"FOR_ITER",
"FUNCTION",
"FUNCTION_PROTOTYPE",
"INCLUDE",
"INITIALIZER",
"PARAMETER",
"PARAMETER_LIST",
"POST_DECR",
"POST_INCR",
"PROGRAM",
"SLIST",
"STRUCT_DECLARATION",
"STRUCT_MEMBERS",
```

```
"SUBSCRIPT",  
"TYPE_MODIFIER",  
"TYPE_SIZEOF",  
"USING_DIRECTIVE"  
};  
}
```

Appendix K

Vita

EDUCATION

Virginia Polytechnic Institute and State University, Blacksburg, VA

Ph.D., Computer Science, 2003

Dissertation: *Implications on the Learning of Programming Through the Implementation of Subsets in Program Development Environments*

Advisor: Dr. John A.N. Lee

M.S., Computer Science, 2000

Villanova University, Villanova, PA

M.S., Computer Science, 1997

Project: *Puma Paint: A Web-Enabled, Interactive Telerobotic Interface*

Advisor: Dr. John A. Lewis

B.S., Computer Science, 1990

RESEARCH INTERESTS

Computer science education and curriculum, educational programming environments, interactive educational tools, software engineering, Internet-based software development, collaborative environments, telerobotic control/interfaces.

PROFESSIONAL EXPERIENCE

Virginia Polytechnic Institute and State University, Blacksburg, VA

Instructor

Computer Literacy, 2000, 1999, 1998

Introduction to Programming in C++, 1999, 2002

Self-study in a Programming Language (Java), 2000, 2001, 2003

Teaching Assistant

Advanced Topics in Software Engineering: Design Patterns, 1999 (graduate course)

Computer Literacy, 1999, 1998

Database Management Systems, 1998 (graduate course)

Introduction to Data Structures and Software Engineering, 1998

Curriculum development for Introduction to Programming in Java (CS1 course), Summer 2003

Villanova University, Villanova, PA

Research Assistant

Ellipsis Resolution in English in Natural Language Processing Systems, developed web-based interface to ellipsis resolution software, 1996

Teaching Assistant

Java programming language, 1997, 1996

e-Nova Solutions L.L.C. / VUSports.com, Blacksburg, VA

Co-owner, Publisher, 1996-present

Manage and publish web site for fans of Villanova athletics.

International Business Machines (IBM), Research Triangle Park, NC

Summer Intern, 2000

Assisted in the development of Enterprise JavaBean support for Visual Age for Java, Enterprise Edition, version 4.0

Sonalysts, Inc., Mt. Holly, NJ

Software Engineer, 1990-1995

Project team leader for developing custom CASE tools, systems administrator (in-house and contracted).

Naval Underwater Warfare Center (NUWC), Newport, RI

Summer Intern, 1989

Responsible for various programming activities and demonstrating simulation systems.

PAPERS \ PANELS

“Subsetting Language Elements in Programming Environments for Novice Students”, Peter J. DePasquale, *6th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2002)*, Orlando, FL, July 14-18, 2002.

“Subsetting Language Elements in Novice Programming Environments”, Peter J. DePasquale, *RESOLVE Programming Language Workshop 2002*, Columbus, OH, June 17-19, 2002.

“An Interactive Program for Determining Junction Depths in Diffused Silicon Devices”, David S. Myers, Jason L. Meyer, Peter J. DePasquale and Robert W. Hendricks, *IEEE Biennial University/Government/Industry Microelectronics (UGIM) Symposium*, Richmond, VA, June 17-20, 2001.

“Let’s commiserate: Advice from students & faculty about making the transition from graduate student to faculty member”, Peter J. DePasquale and K. Todd Stevens, *ACM SIGCSE Technical Symposium on Computer Science Education*, Charlotte, NC, February 22, 2001.

“Teaching using off-the-shelf on-line materials”, Carl Alphonse, Peter J. DePasquale, et. al. *ACM SIGCSE Technical Symposium on Computer Science Education*, Charlotte, NC, February 23, 2001.

“A Java Interface for Asserting Interactive Telerobotic Control”, Peter J. DePasquale, Matthew R. Stein, *SPIE Telemanipulator and Telepresence Technologies IV Conference*, Pittsburgh, PA, October, 1997.

PRESENTATIONS

“Improving Learning of Programming Languages Through the Implementation of Levels in Program Development Environments”, Peter J. DePasquale, *Doctoral Consortium - ACM SIGCSE Technical Symposium on Computer Science Education*, Reno, NV, February 19, 2003.

“Improving Learning of Programming Languages Through the Implementation of Levels in Program Development Environments”, Peter J. DePasquale, *Doctoral Consortium - ACM SIGCSE Technical Symposium on Computer Science Education*, Northern Kentucky/Cincinnati, OH, February 27, 2002.

“Redesigning Programming Language Learning Environments”, Peter J. DePasquale, *Doctoral Consortium - ACM SIGCSE Technical Symposium on Computer Science Education*, Charlotte, NC, February 21, 2001.

“Concurrent Programming Aids”, Peter J. DePasquale, *Doctoral Consortium - ACM SIGCSE Technical Symposium on Computer Science Education*, Austin, TX, March 8, 2000.

“PumaPaint: A Java interface for telerobotic control”, Peter J. DePasquale, Virginia Tech Chapter of the Association for Computing Machinery, Virginia Polytechnic Institute and State University, Blacksburg, VA, March, 1999.

GRANTS

Microsoft software grant of \$99,800, May, 2000

TEXTBOOK ASSISTANCE \ SUPPLEMENTALS

CodeMate content developer, Pearson Education, 2003

Supplemental online materials (*Goin' Live: Build a Web Site*);, *Computer Science Illuminated*, 1st edition, Nell Dale and John A. Lewis, Jones and Bartlett Publishers, 2002, <http://csilluminated.jpup.com/Goin_Live.cfm>.

Supplemental online materials for:, *Java Software Solutions: Foundations of Program Design*, 3rd edition, John A. Lewis, William Loftus, Addison Wesley, 2002.

Instructor's Manuals for:, *The Advantage Series Office XP (Brief, Introductory and Complete editions)*, Sarah E. Hutchinson, Glen J. Coulthard, McGraw-Hill Irwin, 2001.

Supplemental online materials for:, *Java Software Solutions: Foundations of Program Design*, 2nd edition, John A. Lewis, William Loftus, Addison Wesley, 1999.

Appendix O for:, *Java Software Solutions: Foundations of Program Design*, 1st edition, John A. Lewis, William Loftus, Addison Wesley, 1998.

TECHNICAL REPORTS \ ARTICLES

“Implementing OO Designs in EJBs with VisualAge for Java”, Kyle Brown, Peter DePasquale, *IBM VisualAge Developer's Domain*, IBM Corporation, Research Triangle Park, NC, July, 2000

“Emerging Technologies: ATM and Java”, Peter J. DePasquale, Thomas DiSessa, *Concept, Graduate Journal of Interdisciplinary Research*, Villanova University, Villanova, PA, Volume 20, 1997, pp. 21-29.

POSTERS

“The CS1 Sandbox Project: Applying Subsets to a Novice Programmer’s Language”, Peter J. DePasquale, 18th Annual Graduate Student Assembly Research Symposium, Blacksburg, VA, April, 2002.

“The CS1 Sandbox Project: Applying Subsets to a Novice Programmer’s Language”, Peter J. DePasquale, ACM SIGCSE Student Research Poster Contest Presentation, Covington, KY, February, 2002.

“PumaPaint: A Telerobotic Interface”, Peter J. DePasquale, Lara J. Blatchford, ACM SIGCSE Student Research Poster Contest Presentation, San Jose, CA, February, 1997.

PROFESSIONAL ORGANIZATIONS

Association for Computing Machinery (ACM)
The Institute of Electrical and Electronics Engineers, Inc. (IEEE)
ACM Special Interest Group on Computer Science Education (SIGCSE)
IEEE Computer Society

HONORS AND AWARDS

Virginia Polytechnic Institute and State University, Blacksburg, VA

Graduate Award for Teaching Excellence, May, 1999

Villanova University, Villanova, PA

Upsilon Pi Epsilon, May, 1997

SERVICE

Virginia Polytechnic Institute and State University, Blacksburg, VA

Honor Court Panelist, Graduate Honor System, 2000-2003 **Member, System administrator**,
Computer Science department Graduate Council, 1998-1999 **President**, Virginia Tech student chapter
of Upsilon Pi Epsilon, 1999-2000

Villanova University, Villanova, PA

Computer Science department representative, Graduate Student Council, 1996-1997

REFERENCES

Dr. John A.N. (J.A.N.) Lee
Professor (Retired)
Department of Computer Science
Virginia Polytechnic Institute and State University
660 McBryde Hall
Blacksburg, VA 24060
(540) 231-5780
janlee@cs.vt.edu

Dr. Verna Schuetz
Associate Department Head (Retired)
Department of Computer Science
Virginia Polytechnic Institute and State University
660 McBryde Hall
Blacksburg, VA 24060
(540) 231-6932
schuetz@cs.vt.edu

Dr. John Lewis
Associate Professor
Department of Computing Sciences
Villanova University
800 Lancaster Avenue
Villanova, PA 19085
(610) 519-7348
john.lewis@villanova.edu

Dr. Matthew Stein
Assistant Professor
School of Engineering
Roger Williams University
One Old Ferry Road
Bristol, RI 02809
(401) 254-3489
ms@alpha.rwu.edu

This page intentionally left blank.

Index

- .NET, *see* programming environments
- Abstract Syntax Tree, 44, 47, 66
- ACT, *see* scores, ACT
- Ada, *see* programming language
- Advanced Placement test, 73
- ALGOL, *see* programming language
- Alice, *see* programming environments
- American College Test, *see* scores, ACT
- ANTLR, iii, 44–51, 55, 62, 64, 66, 78
 - interpreter
 - grammar input file, 498–523
 - source code output, 524–566
 - lexer
 - grammar input file, 272–277
 - source code output, 278–306
 - parser
 - grammar input file, 307–320
 - source code output, 321–405
 - semantic
 - grammar input file, 406–436
 - source code output, 437–497
- Apache, 62
- Arthur, James, iii
- Association for Computing Machinery (ACM), 1, 2, 35
- AST, *see* Abstract Syntax Tree
- BASIC, *see* programming language
- Bergin, Joe, 2
- Bittitalk, *see* programming environments
- Bjork, Russ, 2
- BlueJ, *see* programming environments
- Brooks, Fredrick, 3
- C, *see* programming language
- C++, *see* programming language
- Chase, Joseph, iii
- COBOL, *see* programming language
- CodeLab, *see* programming environments
- conclusions
 - closed-laboratory, 130–131
 - demographic, 122
 - out-of-laboratory projects, 152–153
- Cornell Program Synthesizer, *see* programming environments
- CS1 Sandbox, 43–66, 76–78, 123, 132, 138
 - AST window, 66
 - data collection, 62–63
 - debug window, 64–65
 - deployment, 56
 - grammar, 259–270
 - graphical user interface, 51–54
 - input/output interface, 51
 - subset creation tool, 57–59
- CS1044, 4, 11, 46, 67, 68, 73, 125
- CS1104, 4, 73
- CS1114, 73
- CS1704, 73
- CS6724, 57, 59
- CSS, 55
- Curator, 50, 71, 79, 81, 82, 86, 87, 130, 133, 148
- CVS, 44
- data
 - academic, 78
 - electronic, 76
 - error recognition experiment, 79
 - focus group, 79
 - survey, 78
- data collection, 76–79
- de Raadt, Michael, 2
- DOS, 18
- DrJava, *see* programming environments
- DrScheme, *see* programming environments
- EDSAC, 3
- Eisenstadt, Marc, 19
- Emile, *see* programming environments
- error, 55
 - code, 55
 - message(s), 55

- Error Recognition Experiment Handout, 245–252
- Euphoria, *see* programming language
- File Transfer Protocol (FTP), 56
- FORTRAN, *see* programming language
- glossary, 55
- Graphical User Interface (GUI), 5, 8, 10
- Gries, David, 37
- Gries, Paul, 37
- Guzdial, Mark, 18
- Hanks, Logan, iii
- help, 54
 - code, 55
 - display system, 54
 - file(s), 54, 55
- Holt Software, 34
- HyperText Markup Language (HTML), 55
- IBM, 22, 34
- Institute of Electrical and Electronics Engineers (IEEE), 1
- Institutional Review Board Application, 71, 171–180
- integrated development environment (IDE), 1
- interpreter, 44, 47–49, 54, 64
- Isenhour, Philip, iii
- Java
 - Run Time Environment, 77
 - Software Development Kit, 29
 - Web Start, 56, 57, 64
- JavaScript, *see* programming language
- jGrasp, *see* programming environments
- JKarelRobot, *see* programming environments
- Kafura, Dennis, 71
- Karel the Robot, *see* programming environments
- Karel++, *see* programming environments
- Kemeny, John, 14
- Khelifi, Ines, 68, 71
- Kurtz, Thomas, 14
- Lee, John A. N. (J.A.N.), iii
- Lewis, John, iii
- lexer, 44, 46–47, 54, 64
- Linux, 62
- Logo, *see* programming language
- Mandrake, 62
- McQuain, William, iii, 67, 73, 93, 130
- Microsoft .NET Common Language Specification, 16
- MiniJava, *see* programming environments
- ML, *see* programming language
- MySQL, 62, 63
- Pérez-Quiñones, Manuel, iii
- parser, 44, 46–48, 54, 64
- Pascal, *see* programming language
- Pascal Trainer, *see* programming environments
- Pattis, Richard, 24
- PHP, 62, 63
- PID, 77, 78, 93
- PL/I, *see* programming language
- ProgramLive, *see* programming environments
- programming environments
 - Alice, 35–36
 - Bittitalk, 22–23
 - BlueJ, 32–34, 148
 - CodeLab, 36–37
 - Cornell Program Synthesizer, 16–17
 - DrJava, 26, 28–29
 - DrScheme, 9, 26–28
 - Emile, 18–19
 - jGrasp, 30–31
 - JKarelRobot, 24
 - Karel the Robot, 1, 24–26
 - Karel++, 3, 24
 - Microsoft .NET, 4, 6, 10, 51, 67, 74, 76, 78, 79, 81, 82, 86, 107, 108, 123, 132
 - Microsoft Visual C++, 4, 7, 10, 51, 74, 123
 - MiniJava, 39–40
 - Pascal Trainer, 17–18
 - ProgramLive, 37–39
 - Ready To Program, 34–35
 - SmallAda, 15
 - Solo, 19–22
 - View Matcher, 22–23
- programming language
 - Ada, 15, 30
 - ALGOL, 14
 - BASIC, 14, 21
 - C, 6, 30, 36
 - C++, 30, 36
 - COBOL, 13–14
 - Euphoria, 1
 - FORTRAN, 14–15
 - Java, 3, 15–16, 24, 26, 28, 30, 32, 34, 36, 37, 39, 40, 43–45, 47–49, 53, 62, 64, 77, 148
 - JavaScript, 36
 - Logo, 1

- ML, 9
- Pascal, 17, 24, 25, 40
- PL/I, 16
- Smalltalk, 22, 23
- Pryor, Jon, iii
- Purdue Compiler Construction Tool Set (PCCTS), iii
- Ready To Program, *see* programming environments
- REPL, 28, 39, 40
- results
 - error recognition experiment, 107, 153–155
 - focus group transcript, 108–116
 - focus group transcript observations, 155–156
 - laboratory
 - compilation/execution attempts, 82–83, 126–128
 - compilation/execution errors, 128–130
 - error frequency, 85
 - scores, 82, 123–126
 - post-project survey, 93–107, 138–150
 - projects
 - compilation/execution attempts, 89, 134–136
 - compilation/execution errors, 136–137
 - error frequency, 90–92
 - scores, 87–88, 132–134
 - time-on-task, 92, 138
 - survey comment observations, 151–152
- Roberts, Eric, 39
- RR Software, Inc., 15
- Sammet, Jean, 13
- SAT, *see* scores, SAT
- Scholastic Aptitude Test, *see* scores, SAT
- Schulman, Robert, iii
- scores
 - ACT, 119
 - programming experience, 119, 121
 - SAT, 119, 121
- semantic validator, 44, 47–49, 54, 64
- SIGCSE, 2, 35
- Slotta, Douglas, iii
- SmallAda, *see* programming environments
- Smalltalk, *see* programming language
- Solo, *see* programming environments
- specifications
 - laboratory, 187–215
 - projects, 217–241
- subset
 - configuration display, 60
 - configuration file, 60, 253–257
 - creation tool, 57–59
 - enforcement, 60–61
- survey
 - demographic, 181
 - post-project, 78, 243–244
- syllabus, 183–186
- test(s)
 - 1-way ANOVA, 119
 - ANOVA, 118, 120–123, 132
 - chi-squared, 117–120
 - Kruskal-Wallis, 118
- Unified Modeling Language (UML), 30, 32
- View Matcher, *see* programming environments
- Virginia Tech Statistical Consulting Center, iii, 117
- Visual C++, *see* programming environments
- Wilkes, Maurice, 3
- WIMP, 5
- Wirth, Niklaus, 9
- XML, 58