SELECTION OF PROGRAMMING LANGUAGES FOR STRUCTURAL ENGINEERING

by

David C. Huxford Jr.

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

MASTER OF SCIENCE

in

Civil Engineering

APPROVED:

S.M. Holzer, Chairman

J.A.N. Lee

R.M. Barker

Y.J. Beliveau

May, 1987
Blacksburg, Virginia

SELECTION OF PROGRAMMING LANGUAGES FOR ENGINEERING

by
David C. Huxford Jr.

Committee Chairman: Dr. Siegfried M. Holzer,
Civil Engineering

(ABSTRACT)

This thesis presents the concepts of structured programming and illustrates how they can be used to develop efficient and reliable programs and aid in language selection. Topics are presented and used to compare several languages with each other rather than with some abstract ideal.

Structured design is a set of concepts that aids the decomposition of a problem using basic block structures into manageable subproblems. Decomposition is a process whereby the large problem is decomposed into components that can be easily understood. This process is continued until the smallest component can be represented by a unit of code performing a single action. By means of the four basic building blocks the atom, concatenation, selection, and repetition one can produce a correct well structured program. In addition, the top-down approach and/or the bottom up approach can assist in producing a structured program that is easy to design, code, debug, modify, and maintain. These approaches minimize the number of bugs and the time spent in the debugging process. Various testing techniques supporting the structured programming process are presented to aid in determining a program's correctness.

The languages must support structured programming. Microsoft FORTRAN, Microsoft QuickBASIC, Turbo Pascal, and Microsoft C are analyzed and compared on the basis of syntactic style, semantic structure, data types and manipulation, application facilities, and application requirements. Example programs are presented to reinforce these concepts. Frame programs are developed in these languages and are used to assist in the language evaluation.

ACKNOWLEDGEMENTS

The author wants to express his thanks to Dr. Siegfried M. Holzer for serving as the authors major advisor, as well as for the ideas, guidance, and advice offered to him throughout his higher educational career.

The author also wishes to express sincere appreciation to Dr. J.A.N. Lee for his expertise, guidance, and thought provoking suggestions during the latter part of the author's research. In addition, th author would like to thank for his aid in coding solutions.

Gratitude is also extended to Dr. Richard M. Barker and to Dr. Yvan J. Beliveau for reviewing this thesis, offering ideas and suggestions, and for serving on the author's committee.

Additional thanks is given to Dr. Kamal Rojiani for his insight, support, and free suggestions on interesting computer topics throughout the author's studies at Va. Tech.

The author would also like to thank Dr. J.M. Duncan, for their support and help during his graduate work.

However, most of all, the author would like to thank his father, for inspiring the author to reach for high goals and for the moral and financial support given to the author during his quest for that knowledge. It is to him that this thesis is dedicated.

Acknowledgements

Lastly, the author would like to thank his fiance', , for the all of the love, support, and laughter during the latter part of the authors education.

to my father.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

INTRODUCTION

This thesis presents the concepts of structured design and structured programming. These concepts are used to produce modular, relatively error free, and reliable programs.

The languages Microsoft QuickBASIC (version 2.01), Microsoft FORTRAN (version 3.3), Turbo Pascal (version 3.0), and Microsoft C (version 4.0) are languages that will be compared for syntactic style, semantic structure, data type and manipulation, application facilities, and application requirements. Structured programming and design concepts are used to determine which of the languages are suitable for structural engineering applications.

Structured programming has become a very important concept in the computing environment. While the computer science field has known about structured design and its benefits for over two decades, many in the engineering field are just beginning to grasp the structured design concepts.

Chapter 1 is dedicated strictly to developing the concepts of structured programming. This thesis is concerned with the development of engineering programs (engineering software) which implies that the whole life of the software will be considered, not just its creation. Thus, this thesis considers the software life cycle. Structured design and structured programming are concepts that support the software life cycle. If both of these concepts are firmly adhered to, the life cycle can be easily and efficiently implemented and supported.

The four basic building blocks -- atom, concatenation, selection, and repetition -- are all that is needed to form a structured program. By using these building blocks, the two principle aims of structured programming, reduction in logical/structure complexity and program correctness are facilitated.

Reduction in complexity can be achieved by using the 'separation of concerns' concept during the decomposition process. The methods of the top-down approach and the bottom-up approach are presented to aid this decomposition process. They will aid in producing a structured program that is easy to design, code, debug, modify and maintain.

The process of ensuring correctness of a program is also discussed. The concepts of verification and program testing are presented. Unit testing, top-down testing, and bottom-up testing are all forms of program testing that can be used. Chapter 1 is concluded by presenting a general outline that can be followed in structured design and the programming process.

Chapter 2 presents the history of the languages to determine their creation intent and standardization. This aids in the determination of the reason of the languages creation.

Syntactic style is discussed in Chapter 3. Syntactic style is a feature of a language that determines the appearance of the code, but it has little effect on the execution. Syntactic style includes the form of identifiers, fixed or free format, statement labels, implicit or declared entities, and program readability.

Identifiers are syntactic objects used to symbolize entities. Fixed or free format refers to the language's sensitivity to blanks, significance of layout, reserved words, method of allowing comments, use of literal strings, and the utilization of the available character set. The ability of a statement label to adequately identify a section of code is discussed. The significance of implicit or declared entities is evaluated. Chapter 3 is completed with a discussion concerning program readability.

Chapter 4 discusses semantic structure. Semantic structures are the features that allow a programmer to form modules to represent an algorithm or data or both. Each of the languages are investigated to determine their support of semantic structure. Semantic structure concerns the control of data and the control of execution.

The control of data refers to the programmers ability to control the representation and domain of data in a program. It also concerns the way in which data is represented, stored and used, as well as the visibility of a variable from different sections of the program. The languages can be classified according to various storage allocation processes. The important concept of variable scope is introduced and further explained by presenting the local, non-local, and global variables. The concepts of external data and data abstraction are also discussed.

Control of execution refers to the language's ability to form the algorithmic structures of a program. The important concepts of blocks and subprograms are discussed. Furthermore, subprograms are further discussed in terms of procedures and functions. The discussion on the control of execution is concluded in with Chapter 4 with the topics of recursion and exception handling.

Data type is discussed in Chapter 5. The basic data types are presented in this chapter. These include the elementary data types of logical, character, numeric (integer, fixed-point, and floating-point), and pointers as well as the aggregate data types of arrays, records, files, and sets. Each data type is discussed, and it's existence in the considered languages is evaluated.

Chapter 6 discusses additional language facilities. Additional language facilities include application facilities, program implementation control, simplicity, standardization, performance, software availability, software development support, mathematical support, chaining, overlays, and concludes with multi-language usage.

Application requirements will be discussed in Chapter 7. This chapter is intended to be a guideline of possible factors to consider in the language selection process. Factors that are discussed include functional operations, size and complexity, expertise, end-user interaction, reliability, timeframe, portability, and memory limitations.

Chapter 8 presents the SOLVE routines used in Holzer's Plane Frame Analysis Program. This routine is developed in the four languages. The execution times, compilation times, and accuracy of each language are determined.

The languages under consideration are evaluated under the criteria discussed throughout Chapters 1-8. In this manner a more in depth study of the languages can occur to determine if they are adequate for structural engineering programs. Conclusions are drawn at the end of each chapter to determine the importance of the topics discussed.

The chapters are organized in such a way as to present the facts in the discussion, while saving the relevancy of those topics for opinions formed in the chapter's conclusion.

STRUCTURED PROGRAMMING

1.0 Introduction

The widespread use of the computer caused changes in the world of programming. Eventually, it was the increasing cost of program construction, the suspect reliability, and the difficulty of modifying outdated programs which led to the development and acceptance of structured design and structured programming [1].

Structured design and structured programming are a part of the software life cycle. The software life cycle consists of several distinct phases: 1) system requirements, 2) software requirements, 3) preliminary design, 4) detailed design, 5) coding and debugging, 6) correctness proving (testing), and 7) maintenance. The life cycle begins with accurate system requirements (i.e., determining what the system should or should not do). After these have been verified (i.e., to determine whether the transcription of data or other operation has been accomplished accurately [2]), software requirements can be determined (i.e., determining in specific terms what the user desires and what input will be required). After these have been verified design can begin. There are two stages to design: 1) preliminary design and 2) detailed design. Each stage should be verified. Only after the design stages can the actual coding (the process of transforming an algorithm into a particular computer language [3]) and debugging (the process of identifying the location and nature of a bug [4]) occur. After coding and debugging, the actual correctness proving of the software system can begin. Once the software has been proven to be correct, it must be maintained [5]. This chapter is concerned only with the design, coding, and testing phases of the life cycle.

Structured design is a set of concepts that aid the decomposition of a problem using basic block structures into manageable subproblems (modules, See section 1.3). These modular units can be obtained by using various structured design techniques. These techniques aid in producing a correct code. There is a general procedure that can be followed to aid structured design and structured programming [6].

1.1 Decomposition

One of the major principles of structured design is the reduction of complexity [7]. The concept of "intellectual manageability" [8] states that a human mind is capable of handling only a few things at one time. Therefore, it is necessary to break a large problem down, or decompose it (as opposed to composition, or building up), into components that can be easily understood [9].

One of the most efficient ways to understand a complicated program (which represents a solution to a problem) is to deal with it in a hierarchical fashion. The program is understood in terms of high level concepts, which are in turn, understood in terms of lower level concepts, and so on. Thus, the original system is decomposed into manageable components (modules). This decomposition must be defined in the program structure where the program components represent the lowest level concepts (See Figure 1.1) [10].

1.2 Control Structure

Control structures are used in the decomposition process. According to the Bohm-Jacopini Theorem there are four basic building blocks needed to construct a structured program [11,12]. If only these four basic building blocks are used throughout a program, the program is theoretically complete [13] and can be proven to be mathematically pure [14]. These basic building blocks are the atom [15], concatenation, selection, and repetition (See Figure 1.2) [16,17,18].

It is important to note that each of these structures allows only one control/logic entrance and only one control/logic exit. If there is more than one entrance or exit then the structure is violated. An imaginary block can be drawn around each structure (thus the term 'block structuring'). Depending on the level of view, this block is called an atom [19]. An atom is considered to be a basic action (See Figure 1.2(a)) [20] and the 'building block' for control structures [21]. At a higher level of consideration, each block can be considered as a single atom by ignoring what occurs inside the dotted line. Thus, an atom can be a single instruction, a sequence of instructions, or a module [22] (See Figures 1.3-1.5). During decomposition, a program can be subdivided into lower level atoms. During composition, blocks can be combined to form higher level atoms (See Figure 1.4) [23].

Atoms can be combined to form three basic control structures. Concatenation results in a sequence of atoms (See Figure 1.2(b)) [24,25]. Selection provides a choice between two or more possibilities (See Figure 1.2(c)) [26,27]. Repetition is represented by looping processes

(See Figure 1.2(d)) [28,29]. It is important to note that each block could, in itself, be any one or series of the control structures and/or modules.

There are three aids that are used to understand a program: enumeration, mathematical induction, and abstraction. Enumeration is the process of verifying "...a property of the computations that can be evoked by an enumerated set of statements performed in sequence, including conditional clauses distinguishing between two or more cases." [30]. Concatenation, which is simply a series of atoms, and selection, which is a branching between a number of atoms, are best understood using enumerative reasoning [31]. Mathematical induction is the process of proving certain statements are true for all cases using mathematics (e.g., the summation of a series) [32]. Repetition, assuming no variance in the processes of the repetitive cycle [33], is usually represented by mathematical induction [34]. Abstraction is the recognition of similarities between objects and the decision to concentrate on these similarities while ignoring the differences [35].

These are only the basic control structures. However, they are not the only ones. Other structures (e.g., case and repeat until (See Figure 1.6) [36]) can be constructed using the four basic building blocks.

1.3 Modularity

A module is "...a general term for a discrete unit of code" [37]. Modular can be defined as constructed for flexibility and variety in use [38]. Modularity evolved over time due to the difficulty of managing monolithic programs. Modularity was not a creation of any one person, but a general consensus of the programming community at that time [39].

The goal of modularity is the decomposition of a program using control structures so that a person can readily understand it. Each module in itself is a control structure or a collection of control structures.

The main characteristic used in developing modules is the concept of 'separation of concerns'. This concept represents the process of the ability to study in depth an aspect of a subject in isolation, while simultaneously knowing that this is only one of the many aspects of the problem. The other aspects will be considered in turn. Thus, the other aspects are not completely ignored, but are temporarily ignored since they are irrelevant to the current topic. This separation is one of the only techniques available for organizing complex problems. By using the 'separation of concerns' concept a structured solution may be developed [40].

There are many methods used to transform an algorithm into modules. These include modular programming, top-down design based on functional decomposition, composite design (Glenford J. Meyers), structured design (Larry L. Constantine), bottom-up design, structured analysis design technique (SADT, Douglas Ross), logical construction of programs (LCP, Jean-Dominique Warnier), and the Jackson design method (Michael Jackson) [41].

Most of these methods are similar in that they incorporate some of the same basic principles with minor modifications. Most of the programming practices used today incorporate some form of the top-down approach (method of successive refinements [42]) and/or the bottom-up approach [43].

1.3.1 Top-Down Approach

The top-down design must be based on a specific coherent definition [44]. Once the definition has been established, the programmer can work down to the details [45]. The importance of a complete, consistent and unambiguous specification must be stressed.

> ...there are...critical problems stemming from a lack of good requirements specification. These include: 1) top-down designing is impossible, for lack of a well-specified 'top'; 2) testing is impossible, because there is nothing to test against; 3) the user is frozen out, because there is no clear statement of what is being produced for him; and 4) management is not in control, as there is no clear statement of what the project team is producing [46].

The top-down approach involves recognizing the important functions to be designed first and continuing from there to the identification of the less important functions that are derived from the major ones. The process involves the breaking down of a large problem into subsets of simpler problems that can be handled individually [47]. The highest level is considered a list of alternative solutions to a problem. According to Chattergy [48], if there is only one alternative, then the problem is not well understood. The problem should be contemplated until there is a list of alternative methods [49]. Choose the best alternative based on available resources [50], ease of implementation, accuracy of the method, memory requirements, and speed of execution [51].

Frame program is a finite element analysis program that was developed using structured design techniques in CE4002. Refer to Figure 1.1. The concept of FRAME PROGRAM would be developed first. FRAME PROGRAM could be broken down into the subprograms of DATA, SYSTEM and RESULT. DATA can further be broken down into STRUCT and LOAD. These, in turn, can be further broken down. This process is repeated until each module represents one function. This diagram can be referred to as a form of a HIPO (i.e., Hierarchical Input, Process, Output) chart [52].

1.3.2 Bottom-Up Approach

The bottom-up approach is the inverse of the top-down approach [53]. In the bottom-up approach, the programmer "...begins with a set of base functions and terminates with a single function implementing the desired program" [54]. The programmer is not concerned with decomposition, but with composition. Each level is composed of lower level functions; each giving way to a more complex function above. The programmer progresses up from the lowest level to the highest level. Thus, the last level designed will call the other levels to produce the desired results of the program [55].

Bottom-up testing goes through unit development first, subsystem development second, and system development last. Bottom-up development usually requires drivers (test harness, test monitor, or test driver). "A test driver must 'exercise' the module under test, in what is basically a primitive simulation of what the superordinate module would do if it were available" [56].

Refer to Figure 1.1 as an example. ASSEMF would be implemented first. With the lowest level implemented, the next level would be implemented with MACT and JLOAD, followed by LOAD for the next level up. With the load subsystem completed, the struct subsystem could be started by implementing CODES, HINGE, MBAND, and PROP. That subsystem would be completed by implementing STRUCT. At this point the subsystem DATA would be implemented. This process would be repeated for the subsystems SYSTEM and RESULT. Lastly, FRAME PROGRAM would be implemented.

1.3.3 Top-Down vs.Bottom-Up

The success of either approach depends upon the programmer's ability to anticipate problems (e.g., coding particulars, algorithm design difficulties, and hardware characteristics) and to generate solutions to avoid them. In practice, neither approach is used exclusively; design proceeds from both top and bottom to meet somewhere in between [57]. The design of a program cannot be accomplished by either method alone, but must involve the iterated application of both methods [58]. The top-down/bottom-up approach is offered only to present a framework in which the design can proceed [59].

The programmer must distinguish between the product and the process that created it. The top-down approach is a nice way to explain and present the final product.

> But who is to say that the process used to construct the product was itself so well structured? Programming involves complex mental processes and a great deal of creativity and ingenuity. No one believes seriously that creativity is structured! To impose a structure on the process by which a programmer works is to limit his creativity: such an imposition is, if strictly enforced, more likely to produce bad programs than good ones [60].

1.3.4 Top-Down coding

The theory of the top-down design is closely related to top-down coding. Coding and design are two separate phases of the life cycle of software (see Figure 1.8). Coding is the process of transforming an algorithm into a particular computer language [61]. Top-down coding is the process of coding in parallel with the various steps in the design of a program [62]. The simplest form of top-down coding suggests not to code until the whole design process (preliminary and detailed) is complete [63]. Coding can be done by one programmer or by a team of programmers. It may help to consider the program as an inverted tree structure, where the top is the main program and the bottom is composed of the smallest functional modules (see Figure 1.1). Programmers should not begin to code the lower level until the previous level of modules is complete (if the lower level consists of library routines, lower level coding will not have to be done [64]). However, programmers should not code another module until the current module has been tested and/or verified [65].

1.3.5 Module Creation

Modules must be independent and highly cohesive (to be united in principles or interests [66]). Independence is the ability of the module to stand alone. Cohesiveness is determined by the number of tasks a module performs. Modules performing one task are more cohesive than modules performing two tasks and so on [67].

Modules are created for a variety of reasons. Modules can provide a linear sequence of execution. They can provide selection of function alternatives (i.e., the selection control structure where logic chooses which module(s) to invoke). They can provide iteration of functions (i.e., the repetitive control structure where each atom is a module). They can aid in distinguishing afferent processes (an afferent module obtains input and delivers it to a higher level [68]), transform processes (a transform module uses input data and modifies it to be output to another module [69]), and efferent processes (an efferent module receives its input from a higher level and outputs it [70]) [71].

When considering an inverted tree structure, modules place calling functions above the called functions, where the higher level function determines what lower level function is to be called and in what sequence. Since modules can be considered to be atoms at a given level of view, they also provide a single entrance and a single exit for each subprogram [72]. Since a module is one small independent part of an overall solution, modules aid in reducing the number of non-trivial errors that may occur in large, non-modularized programs (see Figure 1.9(a)) [73].

Any given module can usually be expressed in terms of smaller modules. Programmers should try to express each module as a single page of code. If he/she cannot, he/she should try to break the module down into a series of submodules [74]. The programmer should consider data design (e.g., parameter passing between modules) as well as algorithm design to ensure that each module has the data that it will need in order to accomplish its task [75], yet not more than it will need [76] (i.e., keep data flow to/from a module minimal).

1.4 Errors and Bugs

The term 'bug' was coined by Captain Grace Hopper in 1945 when she discovered that a moth, which had become lodged in one of the relays of the computer, had caused the computer to stop. Thus, 'bug' became synonymous with the problems and errors associated with computers. The term 'debugging' came to be associated with their solution [77].

In the past years errors and bugs have been considered synonymous. However, this is no longer the case. Bugs and errors are now considered to be two different and distinct entities. A bug is "a mistake or malfunction" [78] where a mistake is "a human action that produces an unintended result" [79] and a malfunction is "the termination of the ability of a functional unit [an entity of hardware,software, or both capable of accomplishing a specified purpose [80]] to perform its required function" [81]. An error is "a discrepancy between computed, observed, or measured value or condition and the true, specified, or theoretically correct value and condition" [82]. Debug has taken a formal definition also: debug is "to detect, to trace and to eliminate mistakes in computer programs..." where mistakes is defined as above [83].

In general, there are two types of bugs and one type of error that may occur. These are syntax bugs, run-time bugs, and logic errors [84]. Syntax bugs violate the basic rules of the language (e.g., spelling, number formation, and line numbering) [85] and are usually detected during the coding/debugging stage of the software life cycle [86]. Run-time bugs are those that occur during the execution of the program (e.g., insufficient number of data entries) [87]. Logic errors are due to faulty logic in the program (e.g., program runs, but produces incorrect results) [88] and are usually not detected until the testing phase of the software life cycle [89].

Bugs and errors must be removed before the program can be released for general use. The process of removing these errors is typically divided up into debugging and testing [90]. As has already been discussed, debugging deals with known mistakes [91]. However, testing is the process of proving program correctness [92].

1.5 Program Correctness

It is hopeless to establish the correctness of a program without taking the structure of the program into account. "The extent to which the program correctness can be established is not purely a function of the program's external specifications and behavior but depends critically upon its internal structure" [93].

Because of its size, a large program requires a high confidence level in the correctness of its individual modules. According to Dijkstra, if the chance of correctness of an individual module is p, the chance of correctness of the whole program, P, composed of N modules, is

$$P=p^N.$$

As N becomes larger, 'p' must become closer and closer to 1.0 if 'P' is to differ appreciably from zero [94].

It is a programmer's task to produce not only a correct program, but to also demonstrate its correctness in a convincing manner [95]. Suppose a program is to implement a function 'f' and is supposed to produce the result $y=f(x)$ given the input data set 'x'. The actually written program 'P' implements the function 'f_p'. Proving correctness is establishing that $f(x)=f_p(x)$ [96].

Two main approaches can be used to show program correctness: 1) "demonstrating, by more or less rigorous mathematical proofs, that the program is consistent with a specification of its intended actions" [97] (i.e., verification, correctness proving, proofs of correctness [98]); and 2) "exercising the program with test data and inspecting the output for deviations from the expected values" [99] (testing, ..."or testing and debugging if error location and correction are also included" [100]). It must be noted that these "...are really two extremes of a spectrum of methods,..." [101].

1.5.1 Verification

Verification proves correctness by a systematic study of the program's specifications and structure [102]. Verification is defined as the methods used to ensure that a program meets its design specifications and that the documentation correctly represents both program and specification [103].

> Program proving (increasingly referred to as program verification) involves expressing the program specifications as a logical proposition, expressing individual program execution statements as logical propositions, expressing program branching as an expansion into separate cases, and performing logical transformations on the propositions in a way which ends by demonstrating the equivalence of the program and its specification [104].

Several things can be done to verify a program. All work should be reviewed prior to compilation. The programmer should use only the basic control structures and increased readability to ensure that the logic of the program is correct. The programmer should study the program module by module to show that each and every module matches the specifications and the structure that is required of it. A program should be reviewed by someone, or by several people, other than the developer so that more bugs may be detected. Verification can be done without running the program, whereas testing cannot [105].

It must be noted, however, that even slightly complex programs are very complicated and time consuming to verify. It has been estimated that one man-month of expert time is required to verify 100 lines of code [106]. The largest program to be proved (as of 1973) was only 2000 statements [107]. Computations involving real data where rounding and truncation occurs are nearly impossible to analyze [108]. "Finally, there is no guarantee that the proof is correct or complete..." [109].

1.5.2 Testing

Program testing is necessary despite all efforts at verification [110]. "Testing seeks to prove correctness by applying inputs X to the program, then checking that the results $f_p(X)$ are identical to the correct ones f(X)" [111]. Furthermore, testing is "the process of demonstrating that a system does what it is supposed to do..." [112].

Considering every possible form of input is impractical due to the infinite number of possible cases [113]. Thus, only a small sample set of input can be tested [114]. Since not all possible input cases can be tested, program testing can be used to discover the presence of bugs, but not their absence [115].

Structural testing involves choosing test cases on the basis of program structure. While a randomly selected set of test cases is statistically insignificant, the selection of test cases based on structure is statistically significant. In any given execution of a program, it is possible that not every statement is encountered. Thus, a test result would not reveal an incorrect statement if that statement is not executed. Therefore, in order to increase the probability of detecting errors through program testing, every statement in the program must be executed at least once during the test process [116] (referred to as the minimal test [117]). The result of the tests in combination with the knowledge of the structure of the program builds the basis of an inductive evidence for the correctness of the software [118].

Test cases should be selected on the basis of the minimal test [119]. Furthermore, every branch, path, and node should be traversed at least once. This implies that every line of code will be executed at least once [120].

"For a[n extremely] large program, exhaustive testing is not feasible. Therefore such programs always contain residual errors that have survived the design, development, and debugging stages. The occurrence of errors in the development of the program..." [121] usually follow a general pattern (see Figure 1.9(b)). The program can become operational once the rate of errors falls below the tolerance of the user [122].

1.5.3 Top-Down Testing

Top-down coding supports top-down testing. Top-down testing requires the use of stubs. A stub (dummy, simulated module [123]) is a module that handles the passing of parameters and allows the programmer to know that the module has been called (see Figure 1.10). A stub should not be actual detailed code. A stub is the next level down in the tree structure, a part that the programmer is not yet concerned about [124].

Top-down testing is the process of verifying and testing the program each time a new module is added in top-down sequence. It may be necessary to add stubs to the program to allow execution. If a bug is detected, it probably lies in the newly added module or in one of the stubs. Once the revised program is verified and tested, another module is added and the process repeated [125]. Thus, beginning with a system of N modules, the $N+1^{th}$ module is added and the behavior of the new system is observed [126]. "If the new combination does not work, the bug may or may not be located in the most recently added module (though frequently that is the case); what is important to us is that something about the new module has aggravated the system to the point where a bug exposed itself..." [127].

Refer to Figure 1.1. FRAME PROGRAM would be designed and tested first with DATA, SYSTEM and RESULT being represented by stubs. DATA would be implemented next with STRUCT and LOAD as stubs. SYSTEM would be implemented with STIFF and SOLVE as stubs. This general process of proceeding down one level and from left to right (the normal convention) would continue until the whole program has been implemented and tested.

It is possible to test a module independently. This is known as the unit test. The phased approach to testing is where each and every module is tested individually, after which all of the modules are combined together to perform a field test [128]. When using this approach, the process of locating the bug(s) is very haphazard [129]. However, subjecting a module to a unit test prior to its incorporation into the program for further testing can aid in alleviating obvious bugs [130].

There are several reasons to use top-down testing. The complexity and time involved in final large system testing is significantly reduced. The major data transfers between modules are tested first. The major bugs are detected first, while the trivial bugs are detected last. It is easier to find the bugs [131] since they will probably (but not necessarily) be located in the most recently added module. Debugging becomes more organized [132] due to the systematic addition of modules. Testing of the program is distributed more evenly through the implementation phase, instead of all testing occurring toward the end of implementation. Modules are tested in their own system environment, thus being allowed to interact with each other as they are developed. Perhaps most importantly, the programmer's morale is improved by seeing results of successful skeleton tests [133]. Top-down testing allows the user to see a preliminary version of the system [134].

1.5.4 Bottom-Up Testing

The misfortune of top-down testing on a top-down implementation is that it is easy to make the same mistakes. Top-down implementations should finally be tested bottom-up [135].

The bottom-up approach consists of unit testing, subsystem testing, and system testing. Test drivers will be needed to accomplish bottom-up testing. The concept of stubs may also be used in bottom-up testing. Refer to Figure 1.1. If MACT is used as a driver, ASSEMF would be implemented first. With LOAD and STRUCT being used as drivers, MACT and CODES would be implemented next, with HINGE, MBAND, PROP and JLOAD as stubs. At this point, DATA would be the driver, while STRUCT or LOAD is implemented, with the other being a stub. Once STRUCT and LOAD are implemented, HINGE, MBAND, PROP, and JLOAD would be implemented one at a time. Meanwhile, similar implementations would be occurring in SYSTEM and RESULT. Lastly, FRAME PROGRAM would be tested. Using test drivers, each and every module would undergo a unit test. Subsystem tests would be next (i.e., Struct, MACT, LOAD, SYSTEM, FORCES and RESULT). The system test would be last (i.e., FRAME PROGRAM) [136].

1.6 Added Extras to Top-Down & Structured Programming

As a result of the modularity and basic control structures, the logic of a program using the top-down design flows from the top to the bottom of the code. This increases the readability and the understandability of the program. The increased readability allows people involved in the verification process to check the program more easily [137].

A GOTO-less implementation is obtained when using only the basic control structures. Since none of the basic control structures allow a GOTO statement, the code is free of the GOTO statement. However, the absence of the GOTO statement does not imply that the program is structured. There are times (e.g., construction of a WHILE-DO loop in FORTRAN77) when a properly placed GOTO statement can make the logic, as well as the structure, of a program understandable (See Figure 1.7) [138].

The modules can be easily modified with little concern for the effects on the other modules. The input and output modules can be altered easily, thus allowing for more versatility. A module can be easily modified by one that is more efficient or faster. As more advances are made in the respective field, additional modules can be modified and/or incorporated into the program. A module can be replaced by another module using the new theory or code [139].

1.7 General Procedure for Structured Program Development

There are several phases in designing a good structured program (See Figure 1.8). These include:

1) Establish system requirements: What the machine itself can and cannot do [140].
2) Establish software requirements: A clear and precise definition of the problem to be solved in terms of data needed and output desired [141].
3) Preliminary and Detailed Design: The actual structured logic of the program can now be specified using top-down design [142].
4) Choice of programming language: Ideally one that has the most suitable structure and data types. (Can be combined with (5)) [143].
5) Coding and Debugging: The actual process of writing the program and identifying the location and nature of a bug [144].
6) Testing: The testing process can begin. Use top-down and bottom-up testing procedures [145].
7) Maintenance: The process of updating the program as more or new information become available [146,147]

Following the above procedure will help keep the programmer one on the right track for producing structured programs [148,149].

1.8 Conclusions

Structured programming is the method of decomposing a specific definition of a solution into modules using the building blocks of the atom, concatenation, selection, and repetition.

The atom is considered to be a single action and is the building block for three control structures. Atoms can be combined to form concatenation, selection and repetition. Concatenation is a series of atoms. Selection is a choice between two or more possibilities.

Repetition is represented by a looping process. Each of these structures allows only one control/logic entrance and one control/logic exit. If only these building blocks are used, the resulting program will be well structured and mathematically pure.

The goal of modularity is the decomposition of a solution using control structures so that a person can readily understand the program. Two methods that are used to achieve modularity through the decomposition process are the top-down approach and the bottom-up approach.

The top-down approach is based on a specific coherent definition. With this definition in mind, the large problem can be broken down into sets of simpler problems that can be solved individually. Thus, the top level of hierarchical functions are designed first while the lowest levels, which are derived from the major ones, are designed last.

The bottom-up approach is the inverse of the top-down approach. The programmer begins with the lower levels and combines them to form the upper levels functions. Thus, through a composition process, the programmer continually combines functions until the program is implemented.

It is important to note that if the lower levels of either of the approaches is in error, then the higher levels will also be in error since they are based on the lower ones. The use of the three control structures will aid in ensuring that the lower levels are correct.

Each of the functions used in the top-down approach and the bottom-up approach can be considered a module. A module is a unit of code. Each module is a control structure or a collection of control structures. A module can provide for a selection of function alternatives as well as for the iteration of functions. Modules place calling functions above called functions, where the higher level function determines what lower level function is to be called and in what sequence.

A bug is mistake or malfunction caused by a human or the failure of some hardware. An error is a discrepancy between the observed value and the theoretical value. Debugging is the process of identifying and removing mistakes from a program. The process of removing errors and bugs is typically divided up into debugging and testing. Debugging concerns known mistakes.

Program correctness is typically accomplished using two basic approaches: verification, and testing. Verification is defined as the methods used to ensure that a program meets its design specifications and that the the documentation correctly represents both program and specification. The verification proof itself may be incorrect, thereby producing incorrect results. In addition, verification is very complex and time consuming.

Testing is the process whereby various data is used as input for the program. The output of that program using that input data is then compared for correctness to the known results of the input data. Testing should at least conform to the minimal test, which guarantees that every line of the code will be executed at least once. Testing is the most practical and commonly used method of demonstrating program correctness.

Three forms of testing support the top-down approach and the bottom-up approach. Unit testing is the process of testing each individual module prior to its incorporation into the program to aid in alleviating obvious bugs. Top-down testing is the process of verifying and testing the program each time a new module is added in the top-down sequence. Stubs will be needed to allow program execution. Bottom-up testing consists of unit tests, subsystem tests, and system tests. Test drivers will be needed to accomplish bottom-up testing.

Testing can be viewed as a form of a Factor of Safety. Testing assures the programmer that the program is producing the correct results. The Factor of Safety increases with the number of tests conducted.

Using structured design an programming techniques produces a program that is easy to design, code, debug, modify, and, most importantly, maintain. These approaches minimize the number of bugs and the time spent in the debugging process. Tests and experiments have been conducted that prove that the time and cost of the program's design, coding, debugging, and testing is significantly decreased. The programmer's morale is increased. The increased readability and understandability of the program yields ease of maintenance and modification. Thus, and engineer can greatly increase the efficiency of program production and maintenance by the use of structured design and structured programming techniques.

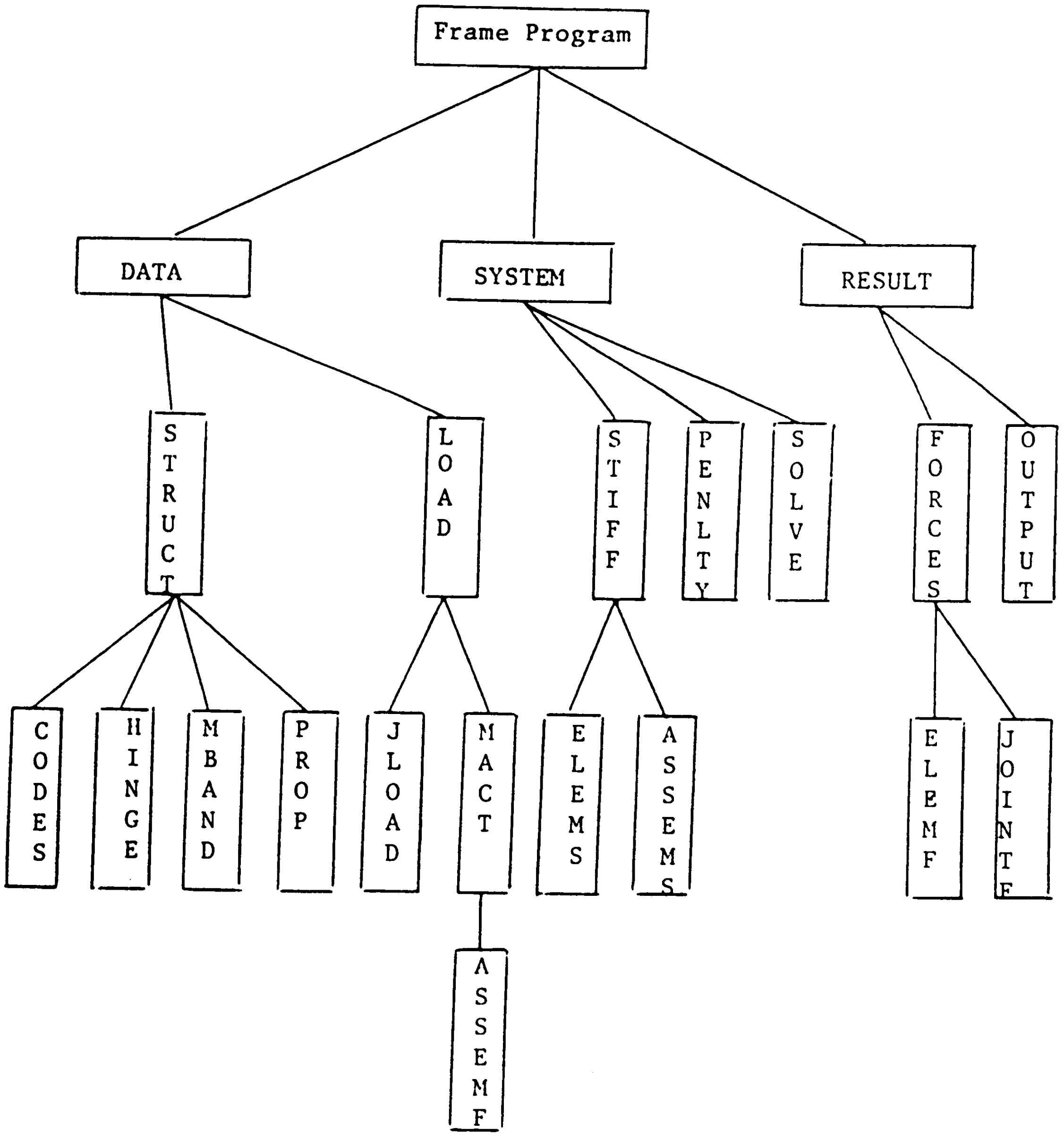
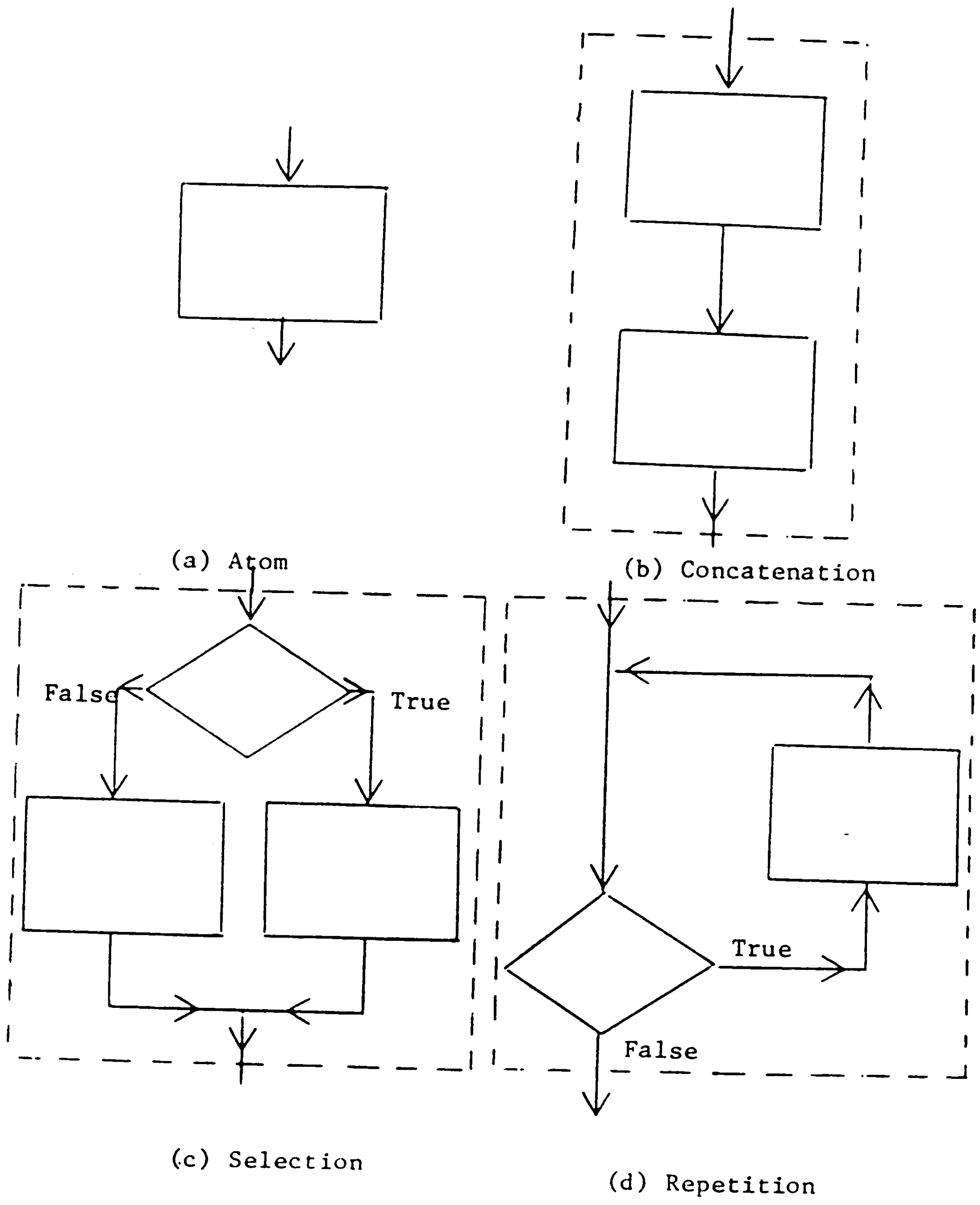


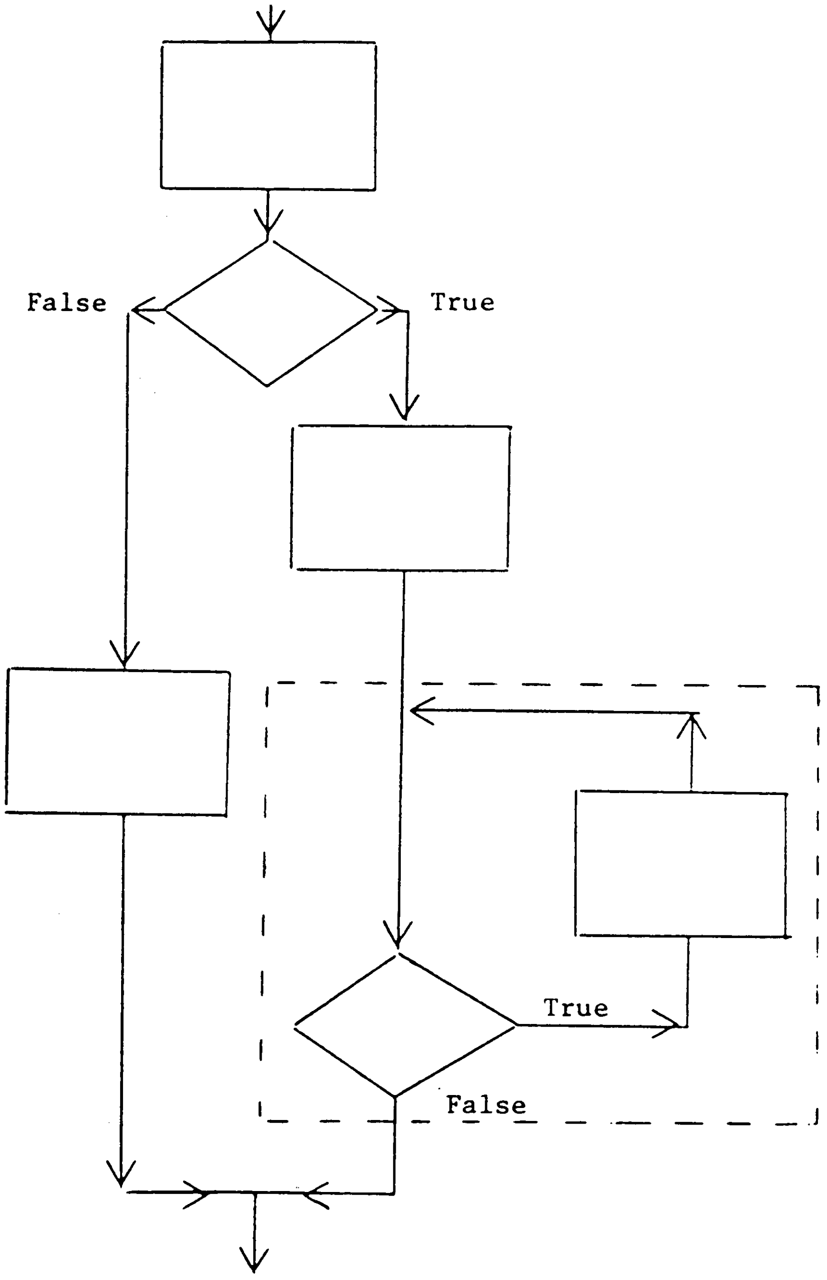
Figure adapted from: Holzer, S.M., Computer Analysis of Structures (New York, New York: Elsevier Science Publishing Company, 1985), p.337.

FIGURE 1.1 - DECOMPOSITION OF FUNCTIONS



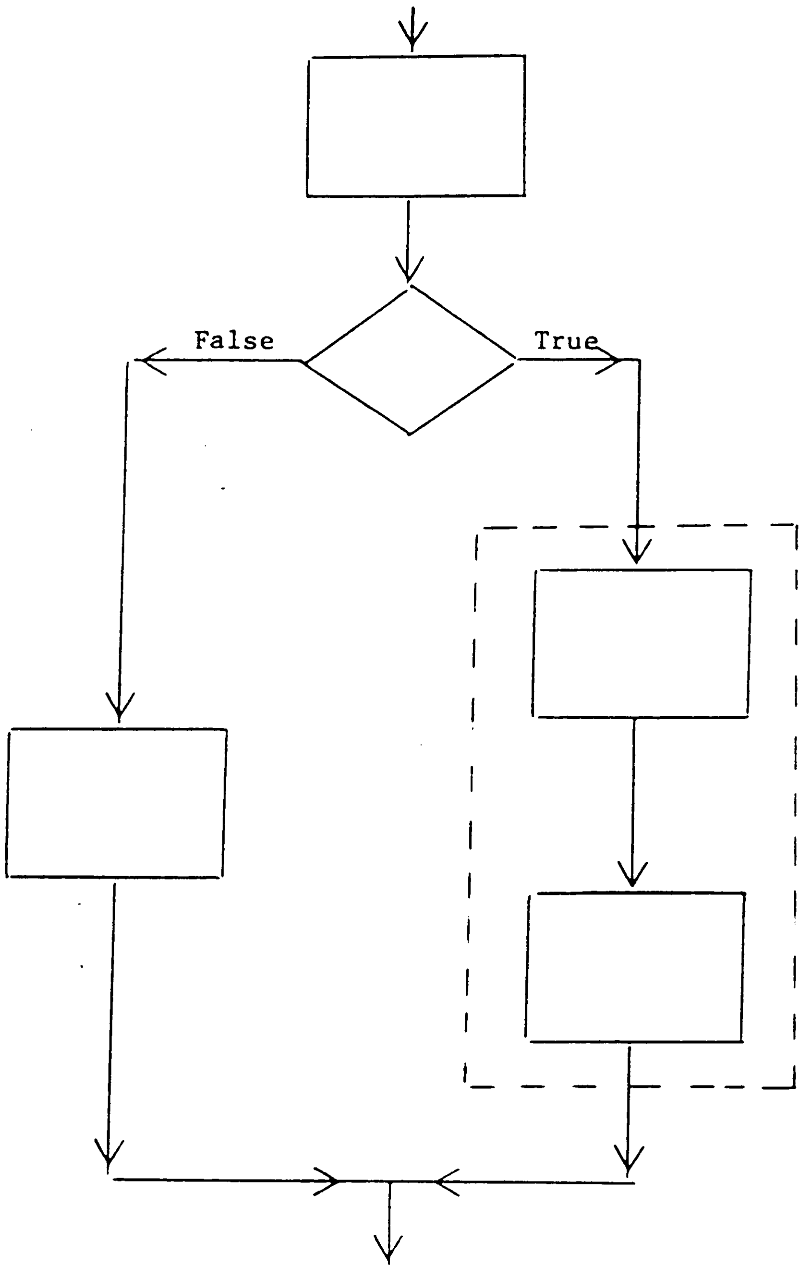
Figures adapted from: Chapra, Steven C. and Raymond P. Canale Introduction to Computing for Engineers (New York, New York: McGraw-Hill, Inc., 1986), p.64.

FIGURE 1.2 - CONTROL STRUCTURES



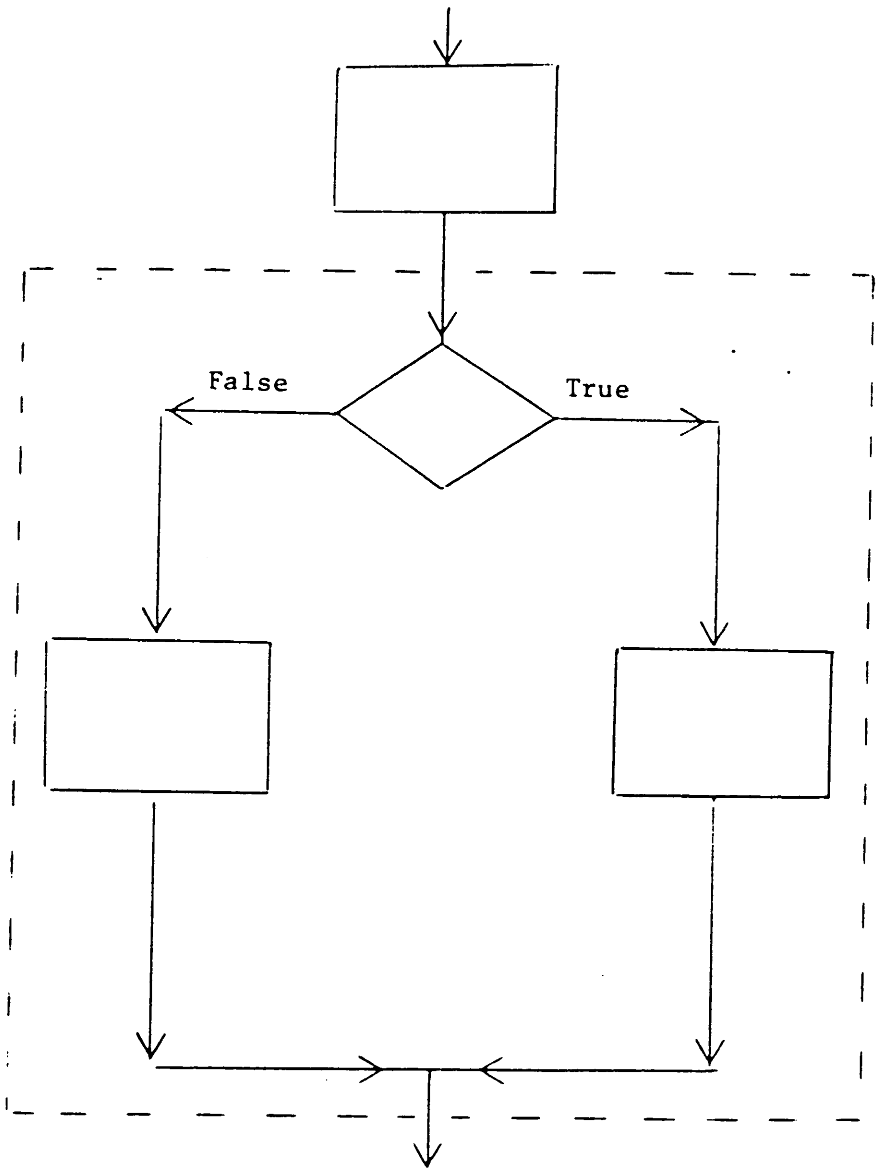
Figures adapted from: Mcgowen, Clement L. and John R. Kelly Top-Down Structured Programming Techniques (New York, New York: Mason/Charter Publishers, Inc., 1975), p.9.

FIGURE 1.3 - REPETITION NESTED IN FLOWCHART



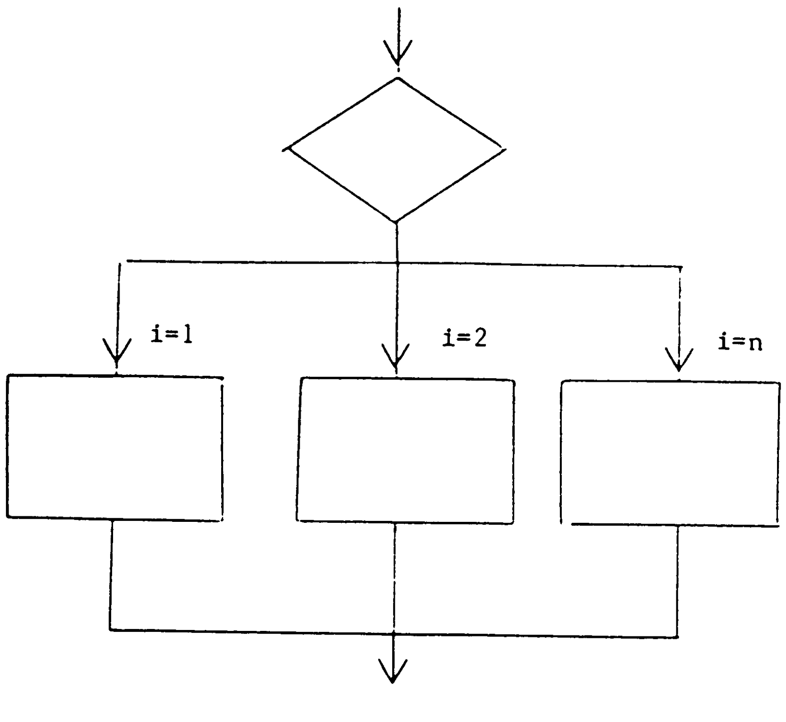
Figures adapted from: Mcgowen, Clement L. and John R. Kelly Top-Down Structured Programming Techniques (New York, New York: Mason/Charter Publishers, Inc., 1975), p.10.

FIGURE 1.4 - CONCATENATION NESTED IN FLOWCHART

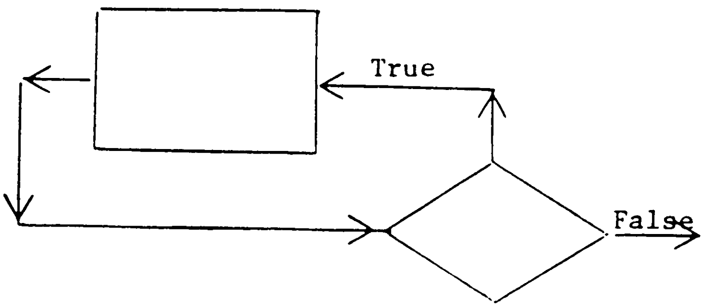


Figures adapted from: Mcgowen, Clement L. and John R. Kelly Top-Down Structured Programming Techniques (New York, New York: Mason/Charter Publishers, Inc., 1975), p.11.

FIGURE 1.5 - SELECTION NESTED IN FLOWCHART



Case



Repeat Until

Figures adapted from: Ledgard, Henry F. and Michael Marcotty "A Geneology of Control Structures" in Communications of the ACM vol.18, no.11 (New York, New York: Association for Computing Machinery, 1975), p.630.

FIGURE 1.6 - HIGHER LEVEL CONSTRUCTS

```
C
C This simulates a while-do loop
C
    READ (5,*) JNUM,JDIR
C
C The while do loop starts here
C
  90 IF (JNUM.NE.0) THEN
       WRITE (6,100) JNUM,JDIR
 100    FORMAT (T8,'*',T23,I4,T58,I4,T75,'*')
       JCODE(JDIR,JNUM)=0
       READ (5,*) JNUM,JDIR
       GOTO 90
C
C The while do loop end here
C
    ELSE
    END IF
```

FIGURE 1.7 - CONSTRUCTIVE USE OF GOTO

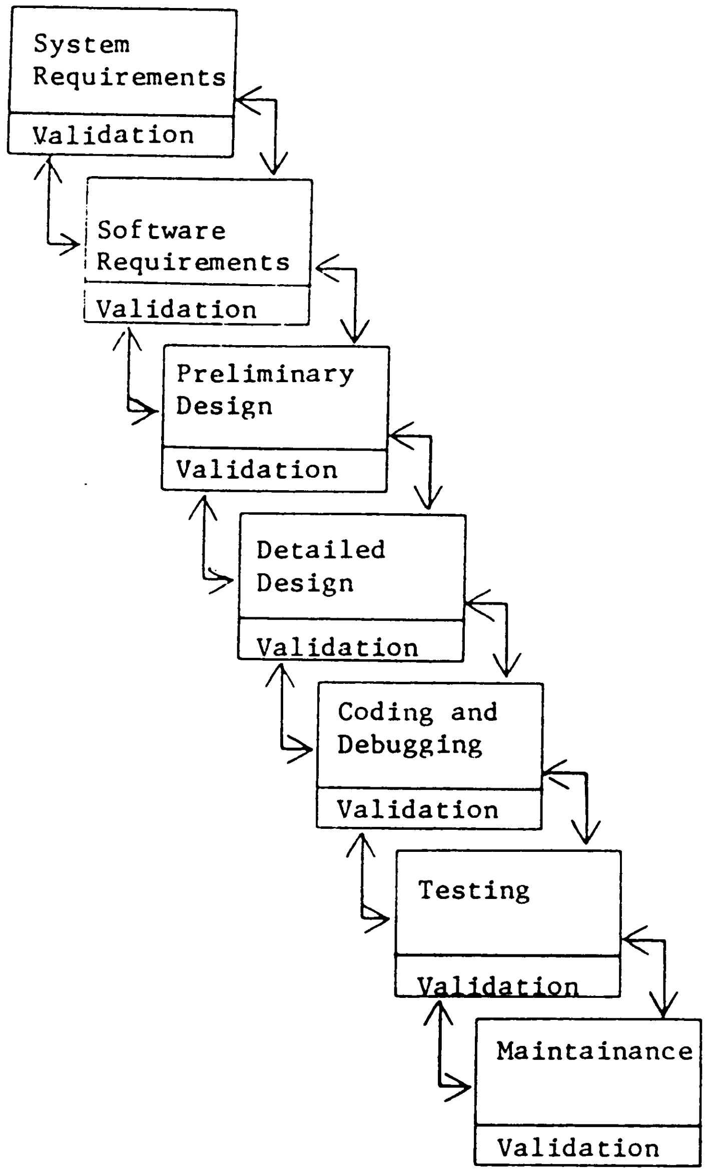


Figure adapted from: Boehm, B.W., "Software Engineering" in IEEE Transactions on Computers c25, no.12 (New York, New York: Institute of Electrical and Electronics Engineers, 1976), p.1227.

FIGURE 1.8 - LIFE CYCLE OF SOFTWARE

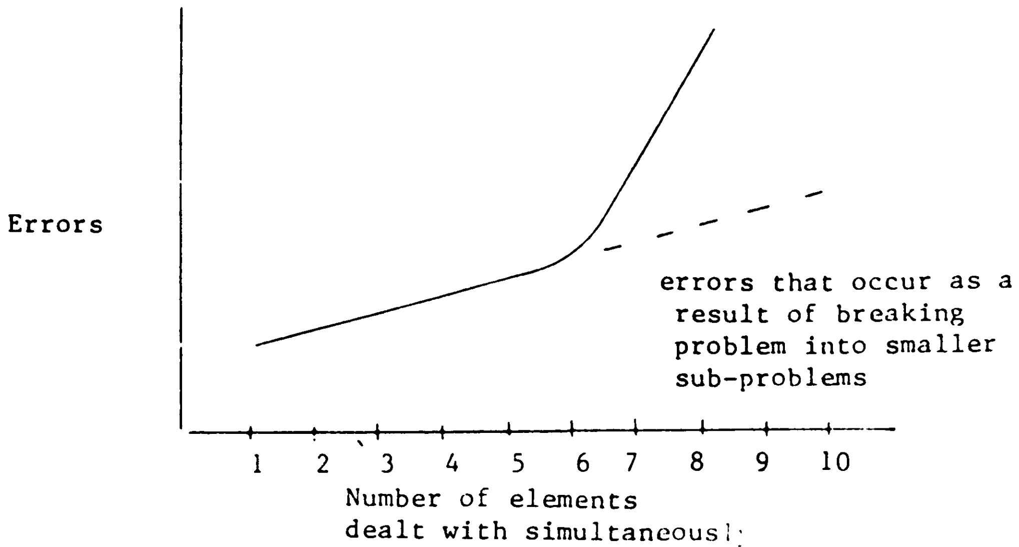


Figure adapted from: Ramamoorthy, C.V. and R.C. Cheung and K.H. Kim, excerpts of "Reliability and Integrity of Large Computer Programs", in Structured Programming, Infotech State of the Art Report (Berkshire, England: Infotech International Limited, 1976), p.103.

Figure 1.9(a)

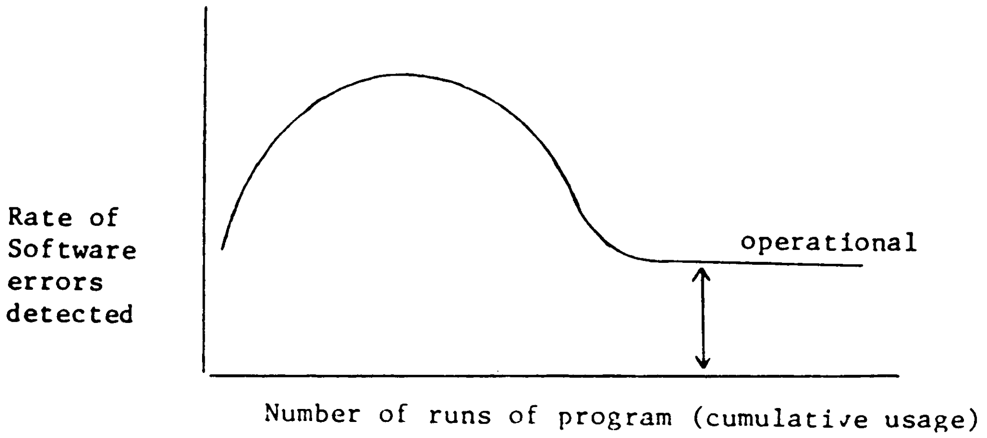


Figure 1.9(b)

Figure adapted from: Yourdon, Edward and Larry L. Constantine STRUCTURED DESIGN: Fundamentals of a Discipline of Computer Program and Systems Design (Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1979), p.71.

FIGURE 1.9 - ERROR CURVES

```
'
' PROGRAM MAIN
'
CHOICE=7
WHILE ((CHOICE<=0) OR (CHOICE>=4))
 CLS
 LOCATE 12,1
 PRINT "This program will branch to a particular subprogram"
 PRINT "   depending upon your choice:"
 PRINT "        1) SUB1"
 PRINT "        2) SUB2"
 PRINT "        3) SUB3"
 INPUT "  Choice:";choice
WEND
IF CHOICE=1 THEN
  CALL SUB1(CHOICE)
ELSE
  IF CHOICE=2 THEN
    CALL SUB2(CHOICE)
  ELSE
    IF CHOICE=3 THEN
      CALL SUB3(CHOICE)
    END IF
  END IF
END IF
END

SUB SUB1(CHOICE) STATIC
' This is a stub
PRINT "SUB1 WAS CALLED"
END SUB

SUB SUB2(CHOICE) STATIC
' This is a stub
PRINT "SUB2 WAS CALLED"
END SUB

SUB SUB3(CHOICE) STATIC
' This is a stub
PRINT "SUB3 WAS CALLED"
END SUB
```

FIGURE 1.10 - QUICKBASIC STUBS

NOTES - CHAPTER 1

1 Perrott, Ronald H., and Donal C.S. Allison, Pascal for FORTRAN Programmers (Rockville, Maryland: Computer Science Press Inc., 1983), p.7.

2 American National Standards Committee X3 American National Dictionary for Information Processing Systems (Washington, D.C.: Computer and Business Equipment Manufacturers Association, 1982), p.143.

3 Lee, J.A.N., Professor of Computer Science, Va. Tech, Personal Interview, 3/2/87.

4 Yourdon, Edward and Larry L. Constantine STRUCTURED DESIGN: Fundamentals of a Discipline of Computer Program and Systems Design (Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1979), p.450.

5 Boehm, Barry W. "Software Engineering", in IEEE Transactions on Computers, C25, No.12 (New York, New York: Institute of Electrical and Electronics Engineers, 1976), p.1227.

6 Swan, Gloria Harrington, Top Down Structured Design Techniques (New York: Petrocelli Books Inc., NY, 1978), pp.x-xii.

7 Bates, D., ed., Structured Programming, Infotech State of the Art Report (Berkshire, England: Infotech International Limited, 1976), p.25.

8 Lee, J.A.N., Personal Interview, 3/2/87.

9 Brown, R.R., "The Technique and Practice of Structured Design ala Constantine," in Infotech State of the Art Report, Volume 2, Invited Papers (Berkshire, England: Infotech International Limited, 1978), pp. 1-23, p. 4.

10 Dahl, O.J., and C.A.R. Hoare, "Hierarchical Program Structures," in Structured Programming (New York, New York: Academic Press Incorporated, 1972), pp.175-220, p 176.

11 Yourdon, Edward, Techniques of Program Structure and Design (Englewood Cliffs, New Jersey: Prentice Hall, 1975), p.146.

12 Ledgard, Henry F. and Michael Marcotty, "A Geneology of Control Structures", Communications of the ACM (Nov. 1975), pp.629-639, p.632.

13 Ledgard, p.632.

14 Dijkstra, E.W., "Notes on Structured Programming," in Structured Programming (New York, New York: Academic Press Incorporated, 1972), pp.1-82, pp. 6-16.

15 Lee, J.A.N., Personal Interview, 3/2/87.

16 Ledgard, p.629.

17 Dijkstra, "Notes on", pp. 16-19.

18 Yourdon, Techniques of Program, p.147.

19 Lee, J.A.N., Personal Interview, 3/2/87.

20 Ledgard, p.629.

21 Holzer, Dr. S.M., Personal Interview, 3/10/87.

22 Dijkstra, "Notes on" pp. 16-19.

23 Lee, J.A.N., Personal Interview, 3/2/87.

24 Dijkstra, "Notes on", p. 17.

25 Linger, R.C. and H.D. Mills and B.I. Witt Structured Programming: Theory and Practice (Reading, Massachusetts: Addison-Wesley Publishing Co., 1979), p.48.

26 Dijkstra, "Notes on", p. 18.

27 Linger, p.52.

28 Dijkstra, "Notes on", p. 19.

29 Linger, p.55.

30 Dijkstra, "Notes on",p. 7.

31 Dijkstra, "Notes on",p. 19.

32 Dijkstra, "Notes on",p. 7.

33 Lee, J.A.N., Personal Interview, 3/2/87.

34 Dijkstra, "Notes on",p. 16-19.

35 Hoare, C.A.R., "Notes on Data Structuring," in Structured Programming (New York, New York: Academic Press Incorporated, 1972), pp.83-174, p. 83.

36 Ledgard, p.630.

37 Microsoft, Microsoft QuickBASIC Reference Manual (Redmond, Washington: Microsoft Corporation, 1986), p.574.

38 Websters New Collegiate Dictionary, p.733.

39 Griffiths, S.N., "Design Methodologies - A Comparison," in Infotech State of the Art Report, Volume 2, Invited Papers (Berkshire, England: Infotech International Limited, 1978), pp.133-166, p 143.

40 Dijkstra, Edsger W., A Discipline of Programming (Englewood Cliffs, New Jersey: Prentice-Hall Incorporated, 1976), p.211.

41 Griffiths, p 143.

42 Chattergy, Rahul, and Udo W. Pooch, Top-Down, Modular Programming in FORTRAN with WATFIV (Cambridge, Massachusetts: Winthrop Publishers Inc., 1980), p.23.

43 Holzer, S.M., Computer Analysis of Structures (New York, New York: Elsevier Science Publishing Company, 1985), p.332.

44 Swan, Gloria Harrington, Top Down Structured Design Techniques (New York: Petrocelli Books Inc., NY, 1978), p.x.

45 Swan, Gloria Harrington, Top Down Structured Design Techniques (New York: Petrocelli Books Inc., NY, 1978), p.x.

46 Boehm, p.1227.

47 Yourdon, Techniques of Program, p.45.

48 Chattergy, p.23.

49 Chattergy, p.23.

50 Lee, J.A.N., Personal Interview, 3/2/87.

51 Chattergy, p.23.

52 Lee, J.A.N., Personal Interview, 3/16/87.

53 Bates, D., ed., Structured Programming, Infotech State of the Art Report (Berkshire, England: Infotech International Limited, 1976), p.87.

54 Denning, P.J., "A Hard Look at Structured Programming," in Structured Programming, Infotech State of the Art Report, ed. D. Bates (Berkshire, England: Infotech International Limited, 1976), p.198.

55 Denning, "A Hard Look at", pp.198-199.

56 Yourdon, STRUCTURED DESIGN:, p.379.

57 Gill, S., Structured Programming, Infotech State of the Art Report, ed. D. Bates (Berkshire, England: Infotech International Limited, 1976), p.92.

58 Bullen, R.H., Structured Programming, Infotech State of the Art Report, ed. D. Bates (Berkshire, England: Infotech International Limited, 1976), p.92.

59 McClure, C.L., Structured Programming, Infotech State of the Art Report, ed. D. Bates (Berkshire, England: Infotech International Limited, 1976), p.96.

60 Denning, p. 200.

61 Lee, J.A.N., Personal Interview, 3/2/87.

62 Yourdon, Techniques of Program, p.54.

63 Yourdon, Techniques of Program, p.54.

64 Lee, J.A.N., Personal Interview, 3/16/87.

65 Yourdon, Techniques of Program, p.54.

66 Websters New Collegiate Dictionary, p.216.

67 Brown, pp.4-5.

68 Yourdon, STRUCTURED DESIGN: , p.445.

69 Yourdon, STRUCTURED DESIGN: , p.464.

70 Yourdon, STRUCTURED DESIGN: , p.451.

71 Brown, p.7

72 Chapin, N., "Function Parsing in Structured Design," in Infotech State of the Art Report, Volume 2, Invited Papers (Berkshire, England: Infotech International Limited, 1978), pp.25-55, p. 24.

73 Yourdon, STRUCTURED DESIGN;, p.71.

74 Yourdon, Techniques of Program, p.49.

75 Yourdon, Techniques of Program, pp.38,45-46

76 Lee, J.A.N., Personal Interview, 3/16/87.

77 Chapra, Steven C. and Raymond P. Canale Introduction to Computing for Engineers (New York, New York: McGraw-Hill, Inc., 1986), p.107.

78 American National Standards Committee X3, p.15.

79 American National Standards Committee X3 , p.83.

80 American National Standards Committee X3 , p.56.

81 American National Standards Committee X3 , pp.79,51.

82 American National Standards Committee X3 , p.48.

83 American National Standards Committee X3 , p.36.

84 Chapra, p.107.

85 Chapra, p.107.

86 Boehm, p.1232.

87 Chapra, p.107.

88 Chapra, p.107.

89 Yourdon, Techniques of Program, p.254.

90 Chapra, p.107.

91 Chapra, p.108.

92 Denning, p.195.

93 Dijkstra, "Notes on",p.5.

94 Dijkstra, "Notes on", p.5.

95 Dijkstra, "Notes on", p.5.

96 Denning, p.195.

97 Bates, p.114.

98 Bates, p.114.

99 Bates, p.114.

100 Bates, p.114.

101 Bates, p.114.

102 Denning, p.195.

103 Bredt, Thomas H., "What the Software Engineer Should Know about Program Verification" in Software Engineering Education, Needs and Objectives, ed. Anthony I. Wasserman and Peter Freeman (New York, New York: Springer-Verlag, 1976), p.28.

104 Boehm, p.1234.

105 Huang, J.C., Structured Programming, Infotech State of the Art Report, ed. D. Bates (Berkshire, England: Infotech International Limited, 1976), p.116.

106 Boehm, p.1235.

107 Boehm, p.1235.

108 Boehm, p.1235.

109 Boehm, p.1235.

110 Bredt, p.33.

111 Denning, p.195.

112 Yourdon, Edward and Larry L. Constantine STRUCTURED DESIGN: Fundamentals of a Discipline of Computer Program and Systems Design (Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1979), p.378.

113 Huang, p.115.

114 Huang, p.115.

115 Huang, p.115.

116 Huang, p.116.

117 Denning, p.196.

118 Kopetz, Software Reliability (Old Woking, Surrey, England: Unwin Brothers Limited, 1979), pp.54-57.

119 Denning, p.196.

120 Huang, p.117.

121 Ramamoorthy, C.V. and R.C. Cheung and K.H. Kim, excerpts of "Reliability and Integrity of Large Computer Programs", in Structured Programming, Infotech State of the Art Report (Berkshire, England: Infotech International Limited, 1976), p.103.

122 Ramamoorthy, p.104.

123 Lee, J.A.N., Personal Interview, 3/2/87.

124 Yourdon, Techniques of Program, pp.59-60.

125 Swann, p.x-xi.

126 Yourdon, STRUCTURED DESIGN:, p.378.

127 Yourdon, STRUCTURED DESIGN:, p.378.

128 Yourdon, STRUCTURED DESIGN;, p.377.

129 Yourdon, STRUCTURED DESIGN;, p.378.

130 Lee, J.A.N., Personal Interview, 3/16/87.

131 Yourdon, Techniques of Program, p.65-70.

132 Yourdon, STRUCTURED DESIGN;, p.378.

133 Yourdon, Techniques of Program, p.65-70.

134 Yourdon, STRUCTURED DESIGN;, p.385.

135 Lee, J.A.N., Personal Interview, 3/2/87.

136 Yourdon, STRUCTURED DESIGN;, p.378.

137 Swann, p.xi.

138 Swan, p.x.

139 Perrott, p.x.

140 Boehm, p.1227.

141 Boehm, p.1227.

142 Boehm, p.1227.

143 Perrott, pp.7-8.

144 Boehm, p.1227.

145 Boehm, p.1227.

146 Boehm, p.1227.

147 Perrott, pp.7-8.

148 Perrott, pp.7-8.

149 Boehm, p.1227.

LANGUAGE HISTORY

2.0 Introduction

Programming was mainly done in assembly or machine language prior to 1954. The emphasis was on the hardware (the computer) rather than the software (the programs). The cost of the hardware was more than the time, effort, and cost of developing software.

The introduction of higher level languages had been attempted before. However, none of these languages lived up to their expectations. This resulted in the misconception that high level languages yielded inefficient programs [1].

High level languages remove the programmer from the machine dependent details of programming. They allow the programmer to program in a pseudo-code that resembles his/her language of communication, not machine code. A high level language allows the programmer to perform a multiplicity of machine dependent tasks by using one statement. For example, the simple assignment statement actually takes several instructions of machine code to carry out. However, with a high level language, it can be done in one instruction. This becomes even more critical when concerned with comparisons (i.e., IF-THEN-ELSE) [2]. Comparisons involve more machine code instructions. Thus, in a high level language, the programmer merely writes 'IF (boolean) THEN (statement) ELSE (statement)'. In machine code this construct would require several more statements instructing the computer where to get values, how to compare them, what to do with the results, and where to resume program execution once the comparison is completed.

By briefly studying the history of the languages being considered, some insight can be obtained as to the reason for their creation and the intention of their use. The standardization of the language will also be considered since 1) "...a standardized language makes it easier and less costly to transport software..." [3]; 2) standardization "...preserves the value of the programmers' skills as they move from one installation to another..." [4]; and 3) standardization provides "...stability to the definition of the language..." [5].

2.1 FORTRAN

In 1954 John Backus of IBM introduced the idea of FORTRAN (FORmula TRANslation). His idea was to introduce a system that would generate programs as efficient as the hand coded machine or assembly programs. He deduced that the key element was the compiler, not the actual language [6]. The compiler was responsible for the task of translating the language into machine code that an individual computer can understand. Thus, the compiler was machine dependent, not the actual language [7].

Backus and his design team originally designed FORTRAN without debugging routines. Despite this, after it was formally announced in 1957, it quickly became popular [8]. FORTRAN was the first high-level language implemented and was used primarily for scientific calculations [9].

After many machine dependent versions were created, standardization of FORTRAN began in 1962. Fortran was standardized in 1966 with FORTRAN66 [10] (two versions of FORTRAN66 were created [11]: FORTRAN and Basic FORTRAN, where Basic FORTRAN was a subset of FORTRAN [12]), thus eliminating most of the machine dependency. In 1978, FORTRAN66 was replaced by a structured FORTRAN, called FORTRAN77, the standardized code that is still used today [13]. While FORTRAN77 is the standardized version, FORTRAN66 is still in use [14]. (For an example listing of FORTRAN77, see Figure 2.1 and Figure 2.5.)

FORTRAN is considered a numerically oriented language. It is most readily applicable to scientific, engineering, and mathematical environments. It is very widely used and accepted [15]. The language is easily available on many machines (including personal computers [16]) and is usually accompanied by a substantial base of software and libraries [17].

2.2 BASIC

Faculty members of Dartmouth College decided that they could develop a language that was easier than FORTRAN [18] while they developing the first time-sharing system. FORTRAN was intended for the experts. BASIC (Beginner's All-Purpose Symbolic Instruction Code [19]) was a simplified language patterned after FORTRAN and ALGOL [20] (for a brief description of ALGOL see Sec 2.3).

The idea was to produce a language that would be adequate for a lay audience. The language should be easy for a beginner to learn. It should be a general language, which can be used to implement any algorithm. Advanced features must be added to make the language easy for the beginner, while attempting not to offend the expert. In addition, the language must 1) take full advantage of the interactive abilities of the computer; 2) give clear and friendly error messages; 3) give fast response for small programs; 4) require no understanding of the hardware; and 5) shield the user from the operating system [21].

The line numbering system was designed strictly for editing. At that time, with no full screen editors, it was convenient to retype the line with the line number and let the computer make the substitution instead of having to retype the whole program [22]. It also facilitated debugging by allowing the programmer to retype the line that the error occurred on instead of retyping the whole program [23].

Dartmouth worked with GE for a while in developing the BASIC language. It wasn't until 1969 when the paths of BASIC started to diverge. In 1972 the joint venture between Dartmouth and GE was dissolved, resulting in different forms of BASIC [24].

Standardization was one way to alleviate the problems plaguing BASIC. However, standardization did not start until 1974, ten years after the invention of BASIC [25]. In 1978 the ANSI (American National Standards Institute [26]) standard for Minimal BASIC was approved after two years of deliberation [27]. The personal computer revolution coupled with the structured programming revolution has changed the way BASIC should be designed [28]. Today another standardization of BASIC is being attempted by the Accredited Standard Committee X3J2, formed in 1974 [29].

A distinction must be made between what is called street BASIC and the structured BASIC that is encouraged today. Street BASIC is the type of BASIC that most people seem to know [30]. These amateur programs are usually made up of spaghetti code (code containing undisciplined use of the GOTO statement; i.e., "Take a listing of Street Basic program and draw a line from each GOTO statement to the target line; the result often resembles a bowl of spaghetti" [31]). The newer versions of BASIC take structure into account, thus making structured BASIC a more desirable programming language [32]. (For an example listing of BASIC, see Figure 2.2 (Proposed Standard) and Figure 2.6 (QuickBASIC)).

BASIC is used for many applications in many environments due to its widespread acceptance, ease of learning, and interpretive mode (most, but not all, installations of BASIC include an interpreter). BASIC's screen control (ability to control the color of the screen and the location of the cursor) and graphics capabilities has made it very useful as a pre-processor and post-processor for many application programs. While it is still misconstrued as an unstructured language, BASIC is widely used and accepted [33].

2.3 Pascal

Pascal was patterned after ALGOL 60. ALGOL 60 is the predecessor of many languages due to its block structure. While ALGOL 60 made little impact in the United States, it was reasonably successful in Europe. However, its successor ALGOL 68 was "...a controversial language in that many people considered it too complex" [34]. Niklaus Wirth of Eidenossische Technische Hochschule (ETH) in Zurich, Switzerland, created Pascal (named after the 17th century mathematician) in the late nineteen sixties.

There were two principal aims of Pascal. One was to make available a language that could be used to teach and enforce structured programming. The other was to develop working forms of this language which were reliable and efficient on the computers of that time [35].

Pascal was also designed with the idea of implementation in mind. Thus the compilers are swift and also produce an executable code that runs quickly. Due to the limited language features, the compilers do not require large amounts of disk space, which makes them ideal for the personal computer [36].

Pascal was designed to be a language of few features. It was stripped of all the unnecessary features. However, these can be reconstructed using the ones that do exist. Thus the language is easy to learn, and quick to compile [37].

Pascal has good debugging facilities [38] (facilities that aid a programmer to "identify a bug's location and nature" [39]) that allow a program to be written in less time with less effort. The strict variable declarations and type checking allow less opportunity to make certain kinds of mistakes. As a result, it takes less time to detect, diagnose, and correct bugs [40]. (For an example listing of Pascal, see Figure 2.3 and Figure 2.7.)

The first version of Pascal was created in 1968. The first compiler was available in 1970. Pascal was standardized in the United States of America in 1977 [41]. An international standard was approved in 1981 [42].

Pascal programs have become extremely portable since its standardization [43]. Many people use Pascal as a teaching tool for structured programming. Various forms of Pascal are available today, all of which meet the requirements of the standardized version. Additional options are also available on a variety of implementations [44].

2.4 C

Ken Thompson, formerly at the University of California at Berkeley, became involved in the MULTICS (MULTiplexed Information and Computer Service [45]) project in the mid nineteen sixties. This was a joint venture between the Massachusetts Institute of Technology, the Honeywell Corporation, and Bell Laboratories. Bell Laboratories eventually dropped from the project [46]. Although MULTICS was an innovative operating system that ran much too slowly on GE machines for many of its intended applications [47], later versions of MULTICS were improved to operate satisfactorily [48].

In his spare time, Ken Thompson put together a single user operating system written in assembly language. He called the system UNIX, which is considered to be a stripped form of MULTICS [49] both in name and in services [50].

Ken Thompson also wrote an interpreter language that he called B. B was similar to BCPL [51], a portable language that made simple assumptions about the machine it operated on. B was chosen as a name of the new language for the first letter of BCPL, which is considered the mother language.

Dennis Ritchie, a cohort in the computer science field, became interested in this interpreter. In 1972 he wrote a modification which he called C, for the second letter of BCPL, not because C follows B in the alphabet. UNIX was then made multi-user and was rewritten in C.

C sacrifices some of its reliability for flexibility. C is more flexible by allowing greater control over machine operations. However, C discards to some extent the reliability controls of a program to accomplish this [52]. C follows the assumption that the programmer knows what he/she is doing. Since C is designed to be small as well as powerful and flexible, many features that would strengthen the reliability of a program are not included (e.g., no check for a situation where a call statement has two arguments, and the subprogram has three arguments) [53].

Although C has not yet been standardized [54], it is expected to be in 1987 or 1988 [55]. However, all of the versions available are similar [56] due to the small size of the C language and the fact that C usually accompanies the UNIX operating system. (For an example of the proposed standard for C, see Figure 2.4 and Figure 2.8.)

Many commercial software companies are re-writing their programs in C [57] due to the programmer's appreciation of the amount of assembly code being generated [58] and the fact that the programmer can constantly rate the program efficiency in terms of machine cycles [59]. C is considered to be a middle level language. It is a step above assembly language and a step or two below BASIC and other higher-level languages [60].

The use of C increases as the number of UNIX operating systems increases. Programmers who need to access the computer at a lower level are choosing C to do so rather than assembly language [61].

2.5 Conclusion

One can begin to see the intended purposes of the languages with a brief review of the computer languages. FORTRAN (1957) was the first successful high level language implemented. BASIC (1964) was initially designed for the lay person to use and understand and to teach him/her the rudiments of programming. Pascal (1968) was designed for teaching and enforcing structured programming methods as well as implementing reliable and efficient programs. C (1976) was designed as a mid-level language to give a programmer access to the hardware while still giving him/her some of the luxuries of a higher level language. FORTRAN, with its broad base of libraries, is so entrenched in today's mathematical and scientific communities that it would be nearly impossible to substitute a new language for it. BASIC is widely used as a first language in lower level academic environments (elementary school and high school). Pascal is widely accepted and used by programmers in many disciplines since they learned it as a teaching language in academic institutions. C tends to be linked with UNIX. As UNIX gains acceptance, so will C.

```
C This program demonstrates concatenation, selection,
C    repetition, and arrays
C
C   LIMIT    CONSTANT, UPPER VALUE OF LOOP
C   DATASET  ARRAY TO BE FILLED AND OUTPUT
C   INDEX    LOOP IDENTIFIER
C
      PARAMETER (LIMIT=10)
      REAL dataset(11)
      INTEGER index

C main;assign values to array, then print out
C   assign values and print out immediately

      DO 10 index=1,10
        dataset(index)=index*index
        IF ((index/2).eq.(index/2.0)) THEN
          write (*,30) 'Value of even array element',
     $                  index,' is ',dataset(index)
        ELSE
          write (*,30) 'Value of odd array element',
     $                index,' is ',dataset(index)
        ENDIF
 10   CONTINUE

C   print out array again, using for loop and constant
      DO 20 index=1,limit
        write (*,30) 'Array element: ',index,
     $               '  Value: ',dataset(index)
 20   CONTINUE
 30   FORMAT (1X,A,I4,A,F7.2)
      STOP
      END
```

Figure adapted from: Brown, Douglas, L. From Pascal to C (Belmont, California: Wadsworth Publishing Company, 1985.), p.61.

FIGURE 2.1 - FORTRAN EXAMPLE

```
10 'this program demonstrates concatenation, selection,
20 '      repetition, and arrays
30  '
40  '    LIMIT    CONSTANT, UPPER VALUE OF LOOP
50  '    DATASET  ARRAY TO BE FILLED AND OUTPUT
60  '    INDEX    LOOP IDENTIFIER
70  '
80  '
90  '
100 '
110 '
120 ' define a constant
130 LIMIT=10
140 '
150 dim dataset%(11)
160 '
170 ' main;assign values to array, then print out
180 '   assign values and print out immediately
190 '
200    FOR index%=1 to 10
210        dataset(index%)=index%*index%
220        IF ((index%/2)=(int(index%/2.0))) THEN
230         print "Value of even array element";index%;
235         print "is ";dataset(index%)
240        ELSE
250         print "Value of odd aray element";index%;
255         print "is ";dataset(index%)
260        END IF
270    NEXT index%
280 '
290 'print out array again, using for loop and constant
300    FOR index%=1 to limit
310      print "Array element: ";index%;"  Value: ";
315      print  dataset(index%)
320    NEXT index%
```

Figure adapted from: Brown, Douglas, L. From Pascal to C (Belmont, California: Wadsworth Publishing Company, 1985.), p.61.

FIGURE 2.2 - PROPOSED STANDARD BASIC EXAMPLE

```
program SAMPLE(input,output);

(*this program demonstrates concatenation, selection,
    repetition, and arrays*)

(*define a constant*)

(*declare global variables*)
var
 index,limit   :integer;
 dataset       :array[1..11] of integer;

(*main;assign values to array, then print out*)
begin
   limit:=10;
   (*assign values and print out immediately*)
   for index:=1 to 10 do
     begin
       dataset[index]:=index*index;
       if ((index/2)=(int(index/2.0))) then
         writeln ('Value of even array element ',
                    index:3,dataset[index]:4)
       else
         writeln ('Value of odd aray element ',index:3,
                   dataset[index]:4)
     end;

 (*print out array again, using for loop and constant*)
   for index:=1 to limit do
     writeln ('Array element: ',index:3,' Value:',
               dataset[index]:4)
end.
```

Figure adapted from: Brown, Douglas, L. From Pascal to C (Belmont, California: Wadsworth Publishing Company, 1985.), p.61.

FIGURE 2.3 - PASCAL EXAMPLE

```
/*this program demonstrates concatenation, selection,
      repetition, and arrays

    LIMIT    CONSTANT, UPPER VALUE OF LOOP
    DATASET  ARRAY TO BE FILLED AND OUTPUT
    INDEX    LOOP IDENTIFIER  */

/*define a constant*/
#define LIMIT 10

/*declare global variables*/
int index;
int dataset[11];

/*main;assign values to array, then print out*/
main()
{
   /*assign values and print out immediately*/
   for (index=1;index<=10;++index)
     {
       dataset[index]=index*index;
       if ((index/2)==(index/2.0))
       {
        printf("Value of even array element %d is
                %d\n",index,dataset[index]);
       }
       else
       {
         printf("Value of odd aray element %d is %d\n",
                index,dataset[index]);
       }
     }

 /*print out array again, using for loop and constant*/
   for(index=1;index<=LIMIT;++index)
     printf("Array element: %d          Value:%d\n",
             index,dataset[index]);
}
```

Figure adapted from: Brown, Douglas, L. From Pascal to C (Belmont, California: Wadsworth Publishing Company, 1985.), p.61.

FIGURE 2.4 - PROPOSED STANDARD C EXAMPLE

```
      SUBROUTINE ASSEMF(F,Q,C1,C2,MCODE,NA,NE)
      DIMENSION
F(6,*),Q(*),C1(*),C2(*),MCODE(6,*),NA(*)
C
C Transform and assemble the local fixed-end forces,
C  F(L,I), to produce the equivalent joint load vector,
C  Q, and the force transformation.
C
C     Statement Function
      FG(C1I,C2I,FLX,FLY)=C1I * FLX + C2I * FLY
C
      DO 20 I=1,NE
       IF (NA(I).NE.0) THEN
         DO 10 L=1,6
          K=MCODE(L,I)
          IF (K.NE.0) THEN
            IF (L.EQ.1) THEN
             Q(K)=Q(K) - FG(C1(I),-C2(I),F(1,I),F(2,I))
            ELSE IF (L.EQ.2) THEN
              Q(K)=Q(K) - FG(C2(I),C1(I),F(1,I),F(2,I))
            ELSE IF (L.EQ.3) THEN
              Q(K)=Q(K)-F(3,I)
            ELSE IF (L.EQ.4) THEN
             Q(K)=Q(K) - FG(C1(I),-C2(I),F(4,I),F(5,I))
            ELSE IF (L.EQ.5) THEN
              Q(K)=Q(K) - FG(C2(I),C1(I),F(1,I),F(2,I))
            ELSE
              Q(K)=Q(K)-F(6,I)
            END IF
          END IF
 10      CONTINUE
       END IF
 20   CONTINUE
      RETURN
      END
```

Figure taken from: Holzer, S.M. Computer Analysis of Structures (New York, New York: Elsevier Science Publishing Company, Inc., 1985), p.349.

FIGURE 2.5 - FORTRAN EXAMPLE, FRAME ANALYSIS SUBPROGRAM

```
 SUB ASSEMF(F(2),Q(1),C1(1),C2(1),MCODE(2),NA(1),NE)
STATIC
'
' TRANSFORM AND ASSEMBLE THE LOCAL FIXED-END-FORCES,
'    F(L,I), TO PRODUCE THE EQUIVALENT JOINT LOAD
VECTOR, Q,
'    AND THE FORCE TRANSFORMATION
'
 PRINT #2, "THIS IS ASSEMF"
 FOR I=1 TO NE
  IF (NA(I) <> 0) THEN
   FOR L=1 TO 6
      K=MCODE(L,I)
      IF (K <> 0) THEN
        IF (L = 1) THEN
           Q(K)=Q(K)-( C1(I)*F(1,I) + (-C2(I)*F(2,I)) )
        ELSEIF (L = 2) THEN
           Q(K)=Q(K)-( C2(I)*F(1,I) + (C1(I)*F(2,I)) )
        ELSEIF (L = 3) THEN
           Q(K)=Q(K)-F(3,I)
        ELSEIF (L = 4) THEN
           Q(K)=Q(K)- ( C1(I)*F(4,I) + (-C2(I)*F(5,I)))
        ELSEIF (L = 5) THEN
           Q(K)=Q(K)-( C2(I)*F(4,I) + (C1(I)*F(5,I)) )
        ELSE
           Q(K)=Q(K)-F(6,I)
        END IF
      ELSE
      END IF
   NEXT L
  ELSE
  END IF
 NEXT I
 END SUB
```

Figure adapted from: Holzer, S.M. Computer Analysis of Structures (New York, New York: Elsevier Science Publishing Company, Inc., 1985), p.349.

FIGURE 2.6 - QUICKBASIC EXAMPLE, FRAME ANALYSIS SUBPROGRAM

```
procedure ASSEMF(f:Force; var q:Loaddisp;
                 c1,c2:Elemprops;
                 mcode:Member; na:Actions;
                 ne:integer);
{
  Transform and assemble the local fixed-end forces,
   f[l,i], to produce the equivalent joint load vector,
   q, and the force transformation.
}
var i,k,l:integer;
begin
for i:=1 to ne do begin
 if na[i] <> 0 then begin
   for l:=1 to 6 do begin
    k:=mcode[l,i];
     if k <> 0 then begin
      case l of
       1: q[k]^:=q[k]^-FG(c1[i]^,-c2[i]^,
                 f[1,i]^,f[2,i]^);
       2: q[k]^:=q[k]^-FG(c2[i]^,c1[i]^,
                 f[1,i]^,f[2,i]^);
       3: q[k]^:=q[k]^-f[3,i]^;
       4: q[k]^:=q[k]^-FG(c1[i]^,-c2[i]^,
                 f[4,i]^,f[5,i]^);
       5: q[k]^:=q[k]^-FG(c2[i]^,c1[i]^,
                 f[4,i]^,f[5,i]^);
       6: q[k]^:=q[k]^-f[6,i]^;
      end;{case}
     end;{if}
   end;{for}
 end;{if}
end;{for}
end;  {of ASSEMF}
```

Figure adapted from: Nuttall, Kenneth R., "Evaluation of Turbo Pascal as a Programming Language for Structural Engineering," Thesis Virginia Polytechnic Institute and State University 1986, pp.141-142.

FIGURE 2.7 - PASCAL EXAMPLE, FRAME ANALYSIS SUBPROGRAM

```
ASSEMF(f,q,c1,c2,mcode,na,ne);
/* Transform and assemble the local fixed-end forces,
      f[l,i], to produce the equivalent joint load
      vector, q, and the force transformation.  */
float *q[],f[];
int i,k,l,c1[],c2[];
{
for (i=1;i<=ne;i++)
{
 if (na[i] != 0)
 {
  for (l=1;l<=6;l++) {
    k=mcode[l,i];
     if (k != 0)
     {
       switch (k)
       {
       case '1':
             q[k]=q[k]-FG(c1[i],-c2[i],f[1,i],f[2,i]);
             break;
       case '2':
             q[k]=q[k]-FG(c2[i],c1[i],f[1,i],f[2,i]);
             break;
       case '3':
             q[k]=q[k]-f[3,i];
             break;
       case '4':
             q[k]=q[k]-FG(c1[i],-c2[i],f[4,i],f[5,i]);
             break;
       case '5':
             q[k]=q[k]-FG(c2[i],c1[i],f[4,i],f[5,i]);
             break;
       case '6':
             q[k]=q[k]-f[6,i];
       } /*switch*/
     } /*if*/
   } /*l for*/
 } /*if*/
} /*i for*/
return(*q)
}  /*of ASSEMF*/
```

Figure adapted from: Holzer, S.M. Computer Analysis of Structures (New York, New York: Elsevier Science Publishing Company, Inc., 1985), p.349.

FIGURE 2.8 - C EXAMPLE, FRAME ANALYSIS SUBPROGRAM

NOTES - CHAPTER 2

1 Perrott, Ronald H., and Donal C.S. Allison, Pascal for Fortran Programmers (Rockville,Maryland: Computer Science Press Inc., 1983), p.1.

2 Huxford, Terry, Former Systems Analyst for Acacia Mutual Life, Personal Interview, 2/9/87.

3 Cugini, John V. Selection and Use of General-Purpose Programming Languages--Overview, National Bureau of Standards Special Publication 500-117, Vol. 1 (Washington D.C.: U.S. Government Printing Office, 1984), p.iii.

4 Cugini, p.iii.

5 Cugini, p.iii.

6 Perrott, p.1.

7 Jones, Jacqueline A., and Keith Harrow, Problem Solving Using Turbo Pascal (Englewood Cliffs, New Jersey: Prentice Hall, 1986), p.xii.

8 Perrott, pp.1-3.

9 Friedman, Frank L., and Elliot B. Koffman, Problem Solving and Structured Programming in FORTRAN (Philippines: Addison-Wesley Publishing Company, 1981), pp.12-13.

10 Perrott, pp.1-3.

11 Lee, J.A.N., Personal Interview, 3/2/87.

12 Perrot, p.2.

13 Perrott, pp.1-3.

14 Lee, J.A.N., Personal Interview, 3/2/87.

15 Schneider, Michael G., and Steven W. Wright, and David M. Perlman, An Introduction to Programming and Problem Solving with Pascal, 2nd ed. (New York, New York: John Wiley & Sons, 1982), p.4.

16 Cugini, p.49.

17 Cugini, p.54.

18 Kemeny, John G. and Thomas E. Kurtz, Back to BASIC, the history, corruption and future of the language (Reading, Massachusetts: Addison-Wesley Publishing Company, Inc., 1985), p.6.

19 Kemeny, p.vii.

20 Kemeny, pp.6-7.

21 Kemeny, pp.6-9.

22 Kemeny, p.11.

23 Kemeny, pp.11.

24 Kemeny, p.23.

25 Kemeny, p.65.

26 Sippi, Charles J. Microcomputer Dictionary 2nd ed. (Indianapolis, Indiana: Howard W. Sams & Co., Inc., 1975) p.19.

27 Kemeny, p.65.

28 Kemeny, p.65.

29 Kemeny, pp.49-50.

30 Kemeny, pp.55-56.

31 Kemeny, p.89.

32 Kemeny, pp.55-56.

33 Kemeny, pp.55-56.

34 Perrot, p.4.

35 Perrott, p.4.

36 Brown, Peter, Pascal from BASIC (Reading, Massachusetts: Addison-Wesley Publishing Company, 1982), p.12.

37 Brown, P., p.12.

38 Perrott, p.x.

39 Yourdon, STRUCTURED DESIGN, p.450.

40 Perrott, p.x.

41 Perrott, pp.4-5.

42 Brown, P., p.13.

43 Perrott, p.x.

44 Foutz, Patricia, Assistant to the Head of the Virginia Tech Computer Science Department, Personal Interview, 1/16/87.

45 Organick, Elliott I. The Multics System: An Examination of Its Structure (Cambridge, Massachusetts: The Massachusetts Institute of Technology, 1972), p.384.

46 Triaster, Robert J., Going from C to BASIC (Englewood Cliffs, New Jersey: Prentice Hall, 1985), pp.4-7.

47 Brown, Douglas L., From Pascal to C (Belmont, California: Wadsworth Incorporated, 1985), p.2.

48 Lee, J.A.N., Personal Interview, 3/2/87.

49 Brown, D., p.6.

50 Lee, J.A.N., Personal Interview, 3/2/87.

51 Brown, D., p.1.

52 Brown, D., p.6.

53 Brown, D., p.6.

54 Brown, D., p.3.

55 Lee, J.A.N., Personal Interview, 3/2/87.

56 Brown, D., p.3.

57 Triaster, Robert J., Going from C to BASIC (Englewood Cliffs, New Jersey: Prentice Hall, 1985), pp.6-7.

58 Traister, p.6.

59 Traister, p.6.

60 Triaster, pp.4-7.

61 Brown, D., p.3.

SYNTACTIC STYLE

3.0 Introduction

Syntax is the way in which symbols are put together to form phrases, clauses, or sentences [1]. "The syntax rules of a language define the form of the language: they define how sentences may be formed as sequences of basic constituents called symbols" [2]. Syntactic style is a feature of a language that determines the appearance of the code, but has little effect on the execution [3]. Cugini suggests syntactic style includes the form of identifiers, fixed or free format (e.g., line ending, indentation), statement labels, implicit or declared entities, and readability [4].

3.1 Identifiers

According to Cugini, "Identifiers (often called names) are syntactic objects used to denote various kinds of entities such as data, type, and procedures" [5].

Only Microsoft FORTRAN restricts the programmer to a small number (six [6]) of characters that can be used to form an identifier. Microsoft FORTRAN will allow more than the number indicated, but those following the first six characters are ignored [7]. Microsoft C deviates from Kernighan and Ritchie by allowing up to 31 characters for the identifier [8]. Turbo Pascal [9] and Microsoft QuickBASIC [10] allow 127 and 40, respectively.

3.2 Fixed Form vs. Free Form

Fixed form and free form refer to the requirements on the form in which statements can be written [11]. "A programming language is greatly affected by the rules that govern its written form" [12]. These include significance of blanks and layout, reserved words, comments, literal strings, and character set [13].

3.2.1 Significance of Blanks

It is common for languages to be insensitive to blanks (e.g., indentation) [14]. Blanks and layout play a significant role in written prose. Ignoring blanks does permit several words to be used as a single identifier (e.g., 'acceleration of gravity') [15]. However, the blank has come to symbolize the termination of a word. Thus, the break character (_) is preferable to the blank (e.g., 'acceleration_of_gravity') [16]. FORTRAN ignores blanks [17]. QuickBASIC [18], Turbo Pascal [19] and C [20] consider blanks as the termination of a word.

Some languages were designed on the assumption that input would be from punched data processing cards [21]. These cards allowed 80 character positions. FORTRAN only considers the first 72 character positions. FORTRAN allows the first character position for comment indicators, or the first five character positions for labels, the sixth character position for a continuation mark, and the other 66 character positions for statements [22]. Character positions 73 to 80 can be used for sequence identification (i.e., to re-assemble the cards correctly if they are out of order by using some code in these character positions [23]). Turbo Pascal, and C allow the start of the statement in any position and require no line numbers [24]. While both BASIC and QuickBASIC allow the start of a statement in any position, QuickBASIC does not require line numbers [25].

3.2.2 Significance of Layout

In most languages, a statement describes a single action to be taken by the computer [26]. The marker that represents the end of the statement is referred to as the statement terminator. Some languages use an implicit terminator (i.e., the end of a physical line; a carriage return,'CR', or a line feed, 'LF'), while others require explicit terminators (i.e., a semi-colon ';'). It is important to note the difference between the statement terminator and the physical end of a line. The statement terminator affects the layout of a program [27]. If the statement terminator is of no significance, a semi-colon is usually used to denote the statement end [28].

```
a:=3.1;
b:=4.2;
c:=5.6; [29]
```

The above example is preferable. However, in certain languages the terminators lack of significance can be taken to an extreme.

```
a
:
 =
 3.1
   ;b:=
  4.2;
 c
:
=
5.6; [30]
```

Since the statement terminator allows the compiler to determine the end of a statement, multiple statements can be placed on one line. While using multiple statements per line is not a good programming practice [31], the following example is given for clarity:

```
a:=3.1;b:=4.2;c:=5.6;
```

QuickBASIC [32] and Microsoft FORTRAN [33] consider the end of a line (i.e., CR, LF, etc.) as the termination of a statement (FORTRAN allows continuation lines [34]). Turbo Pascal [35] and C [36] use a semicolon (;) to separate a statement. Thus, in Turbo Pascal and C it is possible to have several statements per line. More importantly, it is possible to have one statement over several lines [37]. This will increase the readability of the program [38]. Multiple statements per line can be accomplished in QuickBASIC by separating each statement by a ':' [39].

3.2.3 Reserved Words

Most of the languages have a list of reserved words that cannot be used as an identifier [40]. Reserved words limit the range of mnemonic identifiers the programmer may choose from. Microsoft FORTRAN has no reserved words [41] while C [42], Turbo Pascal [43], and QuickBASIC [44] (in increasing order of the number of reserved words) have only a few.

3.2.4 Comments

Comments are characters that are intended for the reader, and are ignored by the compiler [45]. These can be essential to the understanding of an algorithm [46]. FORTRAN requires that comments occupy a line by themselves preceded by the character 'C' or '*'. FORTRAN does not allow inserting comments on lines along with other statements. This practice "...severely restricts [comments] usefulness" [47]. It is advantageous to be able to add comments wherever they are meaningful [48].

```
IF X /*the displacement*/ > 3.754
  THEN CALL ALARM /*because it is
    past the safe limit*/; [49]
```

Pascal [50] and C [51] allow comments as demonstrated above. However, BASIC has the convention of ignoring the rest of a line after the start of a comment. Thus, BASIC allows comments to occur on a line after the code [52,53], but does not allow the comment to be embedded in the statement as was demonstrated above.

3.2.5 Literal Strings

The Hollerith string is still in existence today.

```
18HTHE SQUARE ROOT IS
```

This construction is easier to compile since the compiler does not have to determine the number of characters in the string [54]. That number is already provided (e.g., 18H). "This construction is error-prone, [and] decreases legibility..." [55] than a string delimited by quotes.

```
'The Square Root is'
```

While using quotes is easier for the user [56], the only problem with using quotes is including a quote in a string. Most languages offer solutions to this problem [57] (e.g., using double quotes - "Strings 'can be nested "indefinitely in this way" as you can' see" [58]).

FORTRAN now supports and encourages the use of quotes [59,60], even though it still supports Hollerith strings. QuickBASIC [61], Pascal [62], and C [63] all use quotes (double, single, and double quotes respectively) for string data.

3.2.6 Character Set

Languages are restricted by the number of characters available from input devices [64]. The most commonly used character set is the 64-character punched card set [65]. However, since terminals with an enhanced character set are now the common form of input, FORTRAN tends to abuse the use of characters by not using what is available (e.g., requiring .GT. instead of the character '>') [66]. Most of the other languages make full use of the available characters.

3.3 Statement Labels

Statement labels are a form of identifying a particular section of code. Labels precede the section that it identifies and typically coincide with the first line of code in that section. "Any statement within the program may be labeled, and execution of a GOTO statement causes execution to continue at the referenced statement" [67]. No two statements may have the same label [68].

Labels can be numeric or alphanumeric names. Alphanumeric names are preferable since they can help to describe a section of code [69]. FORTRAN uses numbers as labels [70]. In addition, FORTRAN uses labels to identify the termination of DO-loops [71] and to identify

FORMAT statements [72]. FORTRAN does not allow alphanumeric labels as C does. While standard Pascal requires numeric labels of four digits [73,74], Turbo Pascal allows numeric as well as alphanumeric labels [75]. While standard BASIC requires line numbers for each line (and, thus labels), QuickBASIC does not require line numbers and does allow alphanumeric labels [76].

3.4 Implicit vs. Declared Entities

It is easier to write a program if one can assume the existence of entities (implicit) [77] since they do not have to be declared prior to their use. However, when all variables must be declared prior to referencing them, they are called declared entities [78].

It is easier to assume implicit entities when coding algorithms. In this fashion, identifiers can be introduced without having to return to the beginning of the algorithm to declare their existence. Occasionally the use of implicit entities can lead to unpredictable results due to incorrect assumptions being made. The compiler can no longer check for simple spelling errors in identifiers [79]. An identifier is implicitly declared as a new identifier whenever a misspelled identifier is encountered [80].

The explicit declaration of identifiers alleviates this potentially dangerous problem by allowing strict spelling checks on identifiers. In addition, declared variables do "...a great deal to establish the intent of the program in the reader's mind. The declarations may then be viewed as 'definitions' of program objects, and the executable statements as steps in the process of computing the result" [81].

Microsoft FORTRAN [82] and QuickBASIC [83] allow implicit entities. Turbo Pascal [84] and C [85] force a certain discipline that aids in reading a program by requiring all variables to be declared [86]. C allows only a few declarations implicitly by context [87].

Microsoft FORTRAN [88] and QuickBASIC [89] allow self-typed variables (e.g., FORTRAN's I-N for integer variables and BASIC's $ for alphanumeric string variables). Thus, without declaring them at the beginning of the algorithm, their type is known throughout the program. Microsoft FORTRAN [90] and QuickBASIC [91] allow default typing conventions based on the spelling of the identifier. Microsoft FORTRAN's IMPLICIT statement [92] and QuickBASIC's DEF(type) statement [93] will allow the programmer to define the type of the identifier based on the first letter of that identifier [94].

3.5 Program Readability

"The ability to [create] read[able] programs methodically and accurately is a crucial skill in programming. Program reading is the basis for modifying and validating programs written by others, for selecting and adapting program designs from the literature, and for verifying the correctness of one's own programs." [95].

There are two basic ideas behind program reading:

1) "the verification that a program is correct with respect to a given function" [96]
2) "the determination of the program function of a program" [97].

Program verification has already been discussed (See Sec.1.5.1). The determination of a program function is the process of learning the basic design of the algorithm so that the algorithm can be modified [98].

Documentation is an important part of proper programming. Enough documentation must be provided so that another programmer can modify an algorithm later. If there is not enough documentation, "...the programmer who wishes to modify the program later will not be able to do the job reliably" [99]. Documentation with too much detail, "...usually repeats what is written in the code and serves to obscure rather than enlighten" [100].

A well documented program can be read top-down (See Section 1.3.1) [101]. This process starts with the overall design and concludes with the finest details. One may initially use the declarations and assignment statements and then proceed to verify their expansions later on in the algorithm [102]. However, a poorly documented program must usually be read bottom-up (See Section 1.3.2) [103]. Thus, one must "...discover intermediate abstractions, successively at higher levels, by using those already found" [104]. The reading process can rarely be done strictly top-down or bottom-up. A mixture of the two is usually used [105]. "Readability is thus a much more important criterion that writeability; after all, the program will probably only be written once, but read many times" [106].

The writeability of languages can be described as being either concise or verbose [107]. Concise is defined as "marked by brevity of expression or statement: free from all elaboration and superfluous detail" [108]. Thus, the conciseness of a language will enable a programmer to code an algorithm with little preparation (e.g., declaring labels and all identifiers). Verbose is defined as "containing more words than necessary: impaired by wordiness" [109]. Thus, if a language is verbose it requires a programmer to undergo lengthy preparation to code algorithms.

These two definitions are actually a matter of opinion. For example, a legal document may be considered to be verbose to many people. However, a lawyer may consider the same legal document to be concise in the sense that every word is actually needed [110]. In the same sense, some languages may be verbose to many people. However, to a programmer using structured programming techniques, the same language may be concise in the sense that each statement is needed. For the purposes of this thesis, there must be some clarification of these two abstract terms. A language that does not require variable declarations or other such preparatory statements shall be considered to be concise. A language that does require these preparations shall be considered verbose.

According to Cugini [111], FORTRAN programs tend to be concise, while BASIC, C and Turbo Pascal are verbose. Microsoft FORTRAN's syntax can be very concise (since it was intended for the expert to use):

```
        C Main Program
  DO 10 K=1,10
    KSQUARE=K*K
    WRITE (6,"I2,A,I3") K,' Squared is ',
 $              KSQUARE
          10 CONTINUE
```

QuickBASIC's syntax can be equally concise, yet is usually more understandable (since it was intended for the beginner to use):

```
          Rem Main Program
  FOR K=1 TO 10
    KSQUARE=K*K
    PRINT USING "## Squared is ###";_
```

```
K,KSQUARE
NEXT I
```

Since C and Turbo Pascal are strongly typed languages (i.e., requiring all identifiers to be declared and not allowing any default type declarations), they force more variable declarations, and thus are more verbose, yet more readable (see Figures 3.3 and 3.4).

```
         program main
var
 k,ksquare  : integer;
begin
 for k:=1 to 10 do
 begin
   ksquare=k*k;
   writeln (k:2,' Squared is ',ksquare:3);
          end; (* for k *)
```

Programming in Microsoft FORTRAN and QuickBASIC will produce a running program with little syntactic overhead (program preparation; i.e., declaring identifiers, labels, etc.) (see Figures 3.1 and 3.2). However, care must be taken or the program will quickly become unreadable [112] and unreliable [113]. Turbo Pascal and C tend to be readable, yet they take more work and code to produce even a simple program (see Figure 3.3 and 3.4) [114].

Microsoft FORTRAN and QuickBASIC tend to allow the programmer more freedom to make errors in code development by not requiring declared entities (See Figures 3.1 - 3.4). Turbo Pascal and C require better planning to set up the statements of the language (i.e., Turbo Pascal's BEGIN and END, C's {}, and the declaration of all variables in both Pascal and C). The fact that Turbo Pascal and C are strongly typed languages when compared to Microsoft FORTRAN and QuickBASIC may also add unexpected complications (e.g., unforeseen mixing of data types), thus slowing development, yet increasing readability and reliability.

3.6 Conclusion

Syntactic style is a feature of the language that determines the appearance of the code, but has little effect on the execution. The syntactic style of a language can greatly increase or decrease the readability of a program as well as the ease with which the programmer can make the program readable.

A programmer should have free choice of all possible identifier names. These identifiers should be mnemonic to the entities they represent so as to aid a programmer in remembering what they represent and to increase the readability and understandability of a program. However, Microsoft FORTRAN restricts the programmers choice of identifiers by only considering the first few characters (six) of the identifier. Microsoft C (31 characters) Turbo Pascal (127 characters) and QuickBASIC (40 characters) allow a broader field of choices. Microsoft FORTRAN allows flexibility in identifier choices by not having reserved words while C, Turbo Pascal, and QuickBASIC (in increasing order of the number of reserved words) restrict the programmer in his/her choices.

QuickBASIC, C, and Turbo Pascal consider the blanks to be significant. FORTRAN can interfere with common writing practices, although it hardly ever does, by ignoring blanks during compilation. Microsoft FORTRAN is restrictive by having stern requirements on the placement of code in the field while QuickBASIC, C, and Turbo Pascal allow more freedom in this area. A programmer should have the freedom of placement of code so that readability can be maintained.

In order to increase readability C and Turbo Pascal directly support the ability of a statement to be dispersed over several lines of code. While this is achievable in QuickBASIC and FORTRAN, it is not as directly supported.

A programmer should be able to place comments anywhere within the code in order to maintain readability. FORTRAN is very restrictive in the use of comments by requiring each comment to occupy its own line. QuickBASIC is not as restrictive since it allows comments to occur on the same line as code, but it does not allow the nesting of comments within the code itself. Turbo Pascal and C allow the nesting of comments anywhere within the code.

A programmer should have the option of using mnemonic labels to aid in the readability of a program. QuickBASIC, C, and Turbo Pascal allow mnemonic names for labels to aid in identifying sections of code, while Microsoft FORTRAN requires numbers. In addition, QuickBASIC does not require line numbers for every line as BASIC does. Requiring line numbers for every line confuses the appearance of the code as well as eliminates the use of mnemonic labels since each line number is, in itself, a label.

Structured programming requires that all identifiers be declared prior to their use. Thus, declared entities should always be used, and implicit entities should not ben supported. The increased declarations and preparation can be interpreted as an added factor of safety since it forces the programmer to be absolutely sure of the identifiers he/she is creating. Microsoft

FORTRAN and QuickBASIC allow implicit entities, while Turbo Pascal and C force a certain discipline resulting in increased readability and reliability of an algorithm by using declared entities. Microsoft FORTRAN and QuickBASIC allow the quick development of a program with little syntactic overhead while Turbo Pascal and C force a slower and more reliable development by forcing more preparation.

```
C This program reads and echoes numbers,IN(LIMIT), from
C    INPUT.DAT.  It then determines the minimum (MIN),
C    maximum (MAX), range (RANGE), mean (MEAN),
C    standard deviation (STDEV), and variance
C    (VARIANCE) of the input data and writes it to
C    OUTPUT.DAT.
C  There is a 100 number limit on the input data
      PARAMETER (LIMIT=100)
      REAL MIN,MAX,NUMBER,MEAN,IN(LIMIT)
      OPEN (5,FILE='INPUT.DAT')
      OPEN (6,FILE='OUTPUT.DAT',STATUS='NEW')
      N=1
      SUM=0.0
      TEMP=0.0
C Read and echo numbers
   10 READ (5,*,end=20) IN(N)
         WRITE (6,*) N,' number is ',IN(N)
         N=N+1
      GOTO 10
   20 N=N-1
      MIN=IN(1)
      MAX=IN(1)
C Determine minimum, maximum, and sum of numbers
      DO 30 I=1,N
        IF (IN(I).GT.MAX) THEN
            MAX=IN(I)
        ENDIF
        IF (IN(I).LT.MIN) THEN
            MIN=IN(I)
        ENDIF
   30 SUM=SUM+IN(I)
C Determine mean, variance, st. dev. and output
      MEAN=SUM/N
      DO 40 I=1,N
   40   TEMP=TEMP+((IN(I)-MEAN)**2)
      VARIANCE=(1.0/(N-1.0))*TEMP
      STDEV=SQRT(VARIANCE)
      RANGE=MAX-MIN
      WRITE (6,*) 'Minimum value is ',MIN
      WRITE (6,*) 'Maximum value is ',MAX
      WRITE (6,*) 'Range is ',RANGE
      WRITE (6,*) 'Mean is ',MEAN
      WRITE (6,*) 'Standard Deviation is ',STDEV
      WRITE (6,*) 'Variance is ',VARIANCE
      STOP
      END
```

FIGURE 3.1 - FORTRAN EXAMPLE

```
'  This program reads and echoes numbers,IN(), from
'    INPUT.DAT.  It then determines the minimum (MIN),
'    maximum (MAX), range (RANGE), mean (MEAN),
'    standard deviation (STDEV), and variance
'    (VARIANCE) of the input data and writes it to
'    OUTPUT.DAT.
'  There is a 100 number limit on the input data
LIMIT=100                                  'max numbers input
DIM IN(LIMIT)                              'input numbers
OPEN "INPUT.DAT" FOR INPUT AS #1
OPEN "OUTPUT.DAT" FOR OUTPUT AS #2
N%=1                                       'initialize counter
'               Read and echo numbers until end of file
WHILE NOT EOF(1)
   INPUT #1,IN(N%)
   PRINT USING " ### number is #####.###";N%,IN(N%)
   N%=N%+1
WEND
N%=N%-1          'decrement counter by 1 to get # in file
MIN=IN(1)
MAX=IN(1)
'        Determine minimum, maximum, and sum of numbers
FOR I%=1 TO N%
  IF (IN(I%) > MAX) THEN
      MAX=IN(I%)
  END IF
  IF (IN(I%) < MIN) THEN
      MIN=IN(I%)
  END IF
  SUM=SUM+IN(I%)
NEXT I%
'Determine mean, variance, st. dev. and output
MEAN=SUM/N%
FOR I%=1 TO N%                            'summation of
  TEMP=TEMP+((IN(I%)-MEAN)^2)             '  (X-MEAN)^2
NEXT I%
VARIANCE=(1.0/(N%-1.0))*TEMP
STDEV=SQR(VARIANCE)
RANGE=MAX-MIN
PRINT USING "Minimum value is #####.###";MIN
PRINT USING "Maximum value is #####.###";MAX
PRINT USING "Range is #####.###";RANGE
PRINT USING "Mean is #####.###";MEAN
PRINT USING "Standard Deviation is #####.###";STDEV
PRINT USING "Variance is #####.#####";VARIANCE
END
```

FIGURE 3.2 - QUICKBASIC EXAMPLE

```
(*  This program reads and echoes numbers,IN(), from
    INPUT.DAT.  It then determines the minimum (MIN),
    maximum (MAX), range (RANGE), mean (MEAN),
    standard deviation (STDEV), and variance
    (VARIANCE) of the input data and writes it to
    OUTPUT.DAT.
  There is a 100 number limit on the input data    *)
program reliable;
const
    LIMIT=100;                      (* max numbers input *)
type
    innumber                    = array [1..limit] of real;
var
    infile,outfile               :text;
          innum                  :innumber;
    n,i                          :integer;
    min,max,stdev,variance       :real;
    range,mean,sum,temp          :real;
```

FIGURE 3.3 - PASCAL EXAMPLE

```
begin
   assign(infile,'input.dat');reset(infile);
   assign(outfile,'output.dat');rewrite(outfile);
   N:=1;               (* initialize counter
            Read and echo numbers until end of file *)
   WHILE NOT EOF(infile) do
   begin
      readln(infile,innum[N]);
      writeln(outfile,n:3,' number is ',innum[n]:9:3);
      N:=N+1;
   end;
   N:=N-1; (* decrement count by 1 to get # in file *)
   MIN:=INnum[1];
   MAX:=INnum[1];
   (* Determine minimum, maximum, and sum of numbers *)
   sum:=0.0;
   temp:=0;
   stdev:=0;
   variance:=0;
   FOR I:=1 TO N do
   begin
     IF (INnum[I] > MAX) THEN
         MAX:=INnum[I];
     IF (INnum[I] < MIN) THEN
         MIN:=INnum[I];
     SUM:=SUM+INnum[I];
   end; (*for i*)
   (*  Determine mean, variance, standard deviation and
         range *)
   MEAN:=SUM/N;
   FOR I:=1 TO N do                    (* summation of *)
   begin
     TEMP:=TEMP+sqr(INnum[I]-MEAN);    (*  (X-MEAN)^2 *)
   end;
   VARIANCE:=(1.0/(N-1.0))*TEMP;
   STDEV:=SQRt(VARIANCE);
   RANGE:=MAX-MIN;
   (*                                  Output answers *)
   writeln(outfile,'Minimum value is ',min:9:3);
   writeln(outfile,'Maximum value is ',max:9:3);
   writeln(outfile,'Range is         ',range:9:3);
   writeln(outfile,'Mean is          ',mean:9:3);
   writeln(outfile,'Stand. dev. is   ',stdev:9:3);
   writeln(outfile,'Variance is      ',variance:9:5);
end.
```

FIGURE 3.3 (CONT) - PASCAL EXAMPLE

```
/*  This program reads and echoes numbers,IN(), from
    INPUT.DAT.  It then determines the minimum (MIN),
    maximum (MAX), range (RANGE), mean (MEAN),
    standard deviation (STDEV), and variance
    (VARIANCE) of the input data and writes it to
    OUTPUT.DAT.
  There is a 100 number limit on the input data    */

#define limit 100
```

FIGURE 3.4 - C EXAMPLE

```
main()
{
      int i,n;
      double min,max,stdev,variance;
      double range,mean,sum,temp,in[limit];
      char buffer[80];
      FILE *fin,*fout;
      fout=fopen("output.dat","w");
      fin=fopen("input.dat","r");
        n=1;
        while (fgets(buffer,80,fin)!=NULL)
        {
             in[n]=atof(buffer);
             n++;
        }
        n=n-1;

  min=in[1];
  max=in[1];
  /*  Determine minimum, maximum, and sum of numbers */
  sum=0;
  temp=0;
  stdev=0;
  variance=0;
  for(i=1;i<=n;++i) {
      if (in[i] > max) max=in[i];
      if (in[i] < min) min=in[i];
      fprintf(fout,"%d number is %f\n",i,in[i]);
      sum=sum+in[i];
    }
  /* Determine mean,variance,standard dev., & range */
  mean=sum/n;
  for(i=1;i<=n;++i) {                    /* summation of */
    temp=temp+pow(in[i]-mean,2.0 );       /*(X-MEAN)^2 */
  }
  variance=(1.0/(n-1.0))*temp;
  stdev=sqrt(variance);
  range=max-min;
  /*                                    Output answers */
  fprintf(fout,"Minimum value is %f\n",min);
  fprintf(fout,"Maximum value is %f\n",max);
  fprintf(fout,"Range is         %f\n",range);
  fprintf(fout,"Mean is          %f\n",mean);
  fprintf(fout,"Stand. dev. is   %f\n",stdev);
  fprintf(fout,"Variance is      %f\n",variance);
}
```

FIGURE 3.4 (CONT) - C EXAMPLE

NOTES - CHAPTER 3

1 Webster's New Collegiate Dictionary (Springfield, Massachusetts: G. & C. Merriam Company, 1980), p.1175.

2 Ghezzi, Carlo, and Mehdi Jazayeri Programming Language Concepts 2nd ed. (New York, New York: John Wiley & Sons, 1987), p.42.

3 Cugini, John V. Selection and Use of General-Purpose Programming Languages--Overview, National Bureau of Standards Special Publication 500-117, Vol. 1 (Washington D.C.: U.S. Government Printing Office, 1984), p.4.

4 Huxford, Terry, Personal Phone Interview, 2/3/87.

5 Cugini, p.6.

6 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.28.

7 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.28.

8 Microsoft, Microsoft C Language Reference Manual (Redmond, Washington: Microsoft Corporation, 1986), p.23.

9 Borland, p.43.

10 Microsoft, Microsoft QuickBASIC, p.166.

11 Marcotty, Michael and Henry F. Ledgard Programming Language Landscape (Chicago, Illinois: Science Research Associates, Inc., 1986), p.38.

12 Barron, D.W. An Introduction to the Study of Programming Languages (Cambridge, Great Britain: Cambridge University Press, 1977), p.4.

13 Barron, pp.4-7.

14 Barron, p.4.

15 Barron, p.4.

16 Barron, p.6.

17 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.28.

18 Microsoft, QuickBASIC reference Manual, p.166.

19 Borland, p.43.

20 Kernighan, p.33.

21 Barron, p.6.

22 Merchant, Michael J., FORTRAN77 Language and Style (Belmont, California: Wadsworth Publishing Company, 1981), pp.35-38.

23 Merchant, FORTRAN77 Language and Style, pp.38.

24 Cugini, p.6.

25 Microsoft, QuickBASIC reference Manual, pp.159-160.

26 Borland International, Turbo Pascal Reference Manual (Scotts Valley, California: Borland International, Inc., 1986), p.50.

27 Barron, p.6.

28 Barron, p.6.

29 Barron, p.6.

30 Barron, adapted from p.5.

31 Lee, Personal Interview, 4/15/87.

32 Microsoft, QuickBASIC reference Manual (Redmond, Washington: Microsoft Corporation, 1986), p.158.

33 Microsoft, Microsoft FORTRAN Compiler Reference Manual (Redmond, Washington: Microsoft Corporation, 1986), p.18.

34 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.18.

35 Borland, p.55.

36 Kernighan, Brian W. and Dennis M. Ritchie The C Programming Language (Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1978), p.51.

37 Lee, Personal Interview, 3/2/87.

38 Lee, Personal Interview, 4/15/87.

39 Microsoft, QuickBASIC reference Manual, p.158.

40 Barron, p.6.

41 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.29.

42 Kernighan, p.180.

43 Borland, p.37.

44 Microsoft, QuickBASIC reference Manual, p.525.

45 Barron, p.6.

46 Barron, p.6.

47 Barron, p.6.

48 Barron, p.7.

49 Barron, p.7.

50 Borland, p.46.

51 Kernighan, p.8.

52 International Business Machines Corporation, BASIC Reference (Boca Raton, Florida: International Business Machines Corporation, 1984), p.291.

53 Microsoft, QuickBASIC reference Manual, p.428.

54 Lee, Personal Interview, 4/15/87.

55 Barron, p.7.

56 Lee, Personal Interview, 4/15/87.

57 Barron, p.7.

58 Barron, p.7.

59 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.89.

60 Merchant, FORTRAN77 Language and Style, pp.308,311.

61 Microsoft, QuickBASIC reference Manual, pp.161.

62 Borland, p.68.

63 Kernighan, p.7.

64 Barron, p.7.

65 Barron, p.7.

66 Barron, p.7.

67 Marcotty, p.184.

68 Merchant, FORTRAN77 Language and Style, p.94.

69 Maclennan, Bruce J. Principles of Programming Languages: Design, Evaluation, and Implementation (New York: CBS College Publishing, 1983), p.527.

70 Cugini, p.6.

71 Merchant, p.125.

72 Merchant, p.293.

73 Borland, p.48.

74 American National Standards Institute, and IEEE Standards Board An American National Standard: IEEE Standard Pascal Computer Programming Language (New York, New York: The Institute of Electrical and Electronics Engineers, Inc., 1983), p.27.

75 Borland, p.48.

76 Microsoft, QuickBASIC reference Manual, pp.158-159.

77 Cugini, p.6.

78 Cugini, p.7

79 Marcotty, p.14.

80 Marcotty, p.14.

81 Marcotty, p.14.

82 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.30.

83 Microsoft, QuickBASIC reference Manual, p.166.

84 Borland, p.49.

85 Kernighan, p.36.

86 Cugini, p.7.

87 Kernighan, p.36.

88 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.30.

89 Microsoft, QuickBASIC reference Manual, p.166-167.

90 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.30.

91 Microsoft, QuickBASIC reference Manual, p.248.

92 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.103.

93 Microsoft, QuickBASIC reference Manual, p.248.

94 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.103.

95 Linger, Richard C. and Harlan D. Mills and Bernard I. Witt Structured Programming: Theory and Practice (Reading, Massachusetts: Addison-Wesley Publishing Company, 1979), p.147.

96 Linger, p.148.

97 Linger, p.148.

98 Linger, p.148.

99 Marcotty, p.12.

100 Marcotty, p.12.

101 Linger, p.148.

102 Linger, p.148.

103 Linger, p.148.

104 Linger, p.148.

105 Linger, p.148.

106 Marcotty, p.13.

107 Cugini, p.7.

108 Webster's New Collegiate Dictionary, pp.231-232.

109 Webster's New Collegiate Dictionary, p.1290.

110 Lee, Personal Interview, 4/15/87.

111 Cugini, p.7.

112 Cugini, p.7.

113 Lee, Personal Interview, 4/15/87.

114 Cugini, p.7.

SEMANTIC STRUCTURE

4.0 Introduction

Semantics is the meaning of the language [1]. It is important that a language directly support modules through the meaning of its commands. Thus, semantic structures are the features that allow the programmer to form modules to represent an algorithm or data or both [2]. The importance of modularity has been discussed previously (Modularity, Section 1.3). The two important features of semantic structure are control of data and control abstraction [3] (control of execution [4]).

4.1 Control of Data

This section refers to the programmer's ability to control the representation of data in a program [5] and the way in which data is stored and used as well as the visibility of that data from different sections of the program. The control of data can be divided into storage classes, variable scope, external data, and data abstraction [6].

4.1.1 Storage Classes

A program works with variables, subprograms, statements, and so on. Each of these entities have properties called attributes [7]. These attributes must be defined prior to the use of the entity. "Specifying the exact nature of an attribute is known as binding" [8].

Attributes are things like the name, its type, and the storage area where the value associated with the variable is kept [9]. For example, a program consists of declarations and executables (e.g., statements) [10]. Keywords, pairs, and procedures form declarations [11]. Pairs are made up of identifiers and type [12]. Type consists of integer, boolean, and real [13]. Keywords and operands form executables [14]. Operands are made up of variables and constants [15]. All of these are attributes of entities and must be bound before the entity can be used.

Attributes can be bound at program definition time, at program translation time (compile time), or at program execution time (run-time) [16]. For example, FORTRAN's INTEGER type is bound to a representation at language definition time (i.e., INTEGER must be supported, while the implementation determines what values may be in the type) (See Data Abstraction, Section 4.1.4) [17]. Pascal's type integer may be redefined (i.e., "...the type integer is bound to a representation at translation time" [18]) (See Data Abstraction, Section 4.1.4).

A binding is static if it is defined prior to execution time, and cannot be changed during execution [19]. A binding is dynamic if it is established at execution time, and can be changed (according to language specifications) during execution [20]. "Binding is a central concept in the definition of programming language semantics" [21].

Building on these basic definitions, languages can be classified according to run-time behavior [22] (i.e., how memory is accessed for use). Static languages (e.g., FORTRAN and QuickBASIC) acquire all memory prior to execution [23]. "...All data areas are allocated and arranged by the compiler. By the time the program is loaded into memory, it is ready to run, with only the contents of the memory locations being altered by the program" [24].

Stack-based languages are languages whose "...memory usage is predictable and follows a last-in-first-out [LIFO [25,26]] discipline..." [27]. A stack is "[a] storage device that handles data in such a way that the next item to be retrieved is the most recently stored item still in the storage device..." [28]. Storage will always be added to the top of the stack and taken off of the top of the stack [29]. The programmer only needs to know that data will be accessed in the LIFO format, he/she should not be concerned with the actual storage process. Stack-based languages have various features that "...preclude a static, compile-time layout of memory. Rather, various data areas are allocated and deallocated at run-time by the program" [30]. Thus, stack-based languages have a later binding time than static languages [31]. These languages do not need memory requirements prior to execution [32]. Thus, memory requirements cannot be computed at translation time [33]. The size of the stack is implementation dependent. C and Pascal are basically stack-based languages with some dynamic aspects [34] (although they are not dynamic languages).

In general dynamic languages adopt dynamic rather that static rules [35]. "A dynamic property implies that more of the bindings are carried out at run-time and cannot be done at translation time" [36]. Dynamic languages have unpredictable memory usage [37]. Data can be created at arbitrary points during execution [38] (e.g., the flexible array - an array whose bounds vary during execution). "...Dynamic variables denote data objects whose size and/or number can vary dynamically during their lifetimes" [39]. In addition, some dynamic languages support dynamic typing. Dynamic type implies that "...the type of the variable and therefore the method of access or even the allowable operations cannot be determined at translation time" [40]. None of the languages under consideration are dynamic.

4.1.2 Variable Scope

"The scope of a name [identifier] is the part of the program over which the name is defined" [41]. "The original intent of scope rules was to allow localization of names and to permit the reuse of names in other contexts" [42].

Consider the basic structure of concatenation. This is a series of blocks, where each block can, in itself, be any of the basic structures or series of structures. The identifiers within any one of these blocks must have the capability of being local. Local is defined as "[p]ertaining to that which is defined and used only in one [block] of a computer program" [43], or the identifier "...is only visible (known) within a single [subprogram]" [44]. A local variable has meaning only while the functioning block in which it is defined is being executed [45].

The data within these blocks may be non-local, but the option for the data to be local must exist [46]. Non-local is defined as "[p]ertaining to that which is defined in one [block] of a computer program and used in at least one other [block] of that computer program" [47], or the identifier can be "...used in more than one program unit (a subroutine, function, or the main program) and still refer to the same entity." [48]. There is a subtle difference between non-local and global variables. "Non-local variables that can be referenced by every unit in the program are called global variables" [49].

"Static scope binding defines the scope of a variable in terms of the lexical structure of a program, that is, each reference to a variable is statically bound to a particular (implicit or explicit) variable declaration" [50]. The rules for static scope binding (lexical scoping [51]) are:

- The scope of a declaration includes the block in which it occurs, but excludes any block surrounding it.
- The scope of a declaration includes any block contained within the block in which the declaration occurs but excludes any contained block in which the same identifier is redeclared [52].

"One effect of these rules is to prevent access to variables and procedures declared within a procedure from outside the procedure" [53] (See Figure 4.1a). In addition, if the identifier is redeclared in a nested procedure, the value of the original entity cannot be referenced in that procedure, but can be in the outer procedure [54] (See Figure 4.1b).

Local variables aid in making subprograms "...a truly autonomous entity" [55]. Changing the value of a local variable will not effect the variables outside that block. Local variables permit efficient use of storage since the variable exists only as long as the procedure is invoked [56] (See Figures 4.2 and 4.3). The use of local variables will decrease the chance of unforeseen 'side effects' that may occur when non-local variables are used.

The use of non-local variables can considerably increase the complexity of a program by forcing the programmer to consider how the non-local variable is used throughout the whole program instead of just within the block alone as with local variables [57]. Global variables can be accessed by any part of the program [58]. Thus, caution must be used in the choice of identifiers so that changes made to one identifier will not inadvertently effect an identifier of the same name in another part of the program [59] (See Figure 4.4 and 4.5).

4.1.3 External Data

At times it is advantageous to be able to declare some variable(s) as external [60]. These variables are global in scope [61], and, therefore, are accessible by all program units (See Subprograms, Section 4.2.2) [62,63] inside their nesting and after their declaration [64]. Some languages (e.g., C) treat external data as a global variable that remains in existence permanently [65]. Thus, it will retain its value even after the function is completed [66]. The use of external data allows subprograms easy access to data without the passing of parameters and also allows the use of overlays (See Section 6.7) [67]. External data is not considered a good programming practice, and, therefore, should not be used.

C [68], FORTRAN [69,70,71,72] (e.g., the COMMON statement), and QuickBASIC [73] (e.g., the SHARED statement) support external data. FORTRAN [74,75] and QuickBASIC [76] require that the COMMON and SHARED statements appear in every subprogram that is intended to use these variables. Since Pascal does not allow external procedures, it has no provisions for external data [77].

4.1.4 Data Abstraction

Data abstraction is the concern of disassociating a data type (e.g., integer, real, boolean, etc.) from how it is stored in the machine. The actual stored value is a series of bits (a bit is what is used to store data in a computer and is defined as follows: "In pure binary numeration system, either of the digits 0 and 1" [78]. FORTRAN was one of the first languages to interpret these memory locations "...not as a sequence of anonymous bits, but as an integer value, a real, a boolean, or something else" [79].

> Notice that all of the numeric operations in FORTRAN are representation independent, that is, they depend on the logical, or abstract, properties of the data values not on the details of their representation on a particular machine. A set of values, together with a set of operations on those values that is defined without reference to the representation of the data values, is called an abstract data type [80].

There can be several levels of data abstraction. One of the most basic is to allow the programmer to assign a data type (e.g., integer, float(6), real*8, complex, logical, character, etc.) to an identifier and use the identifier from then on to further define data items [81]. Thus, "...data items are described in terms of their logical meaning, rather than their physical implementation (VELOCITY as opposed to FLOAT(6))" [82]. Modifications can be easily made by altering the initial assignment instead of being forced to effect changes throughout the program if that initial assignment had not been used [83]. C is the only language under consideration that allows the assignment of a data type to an identifier [84,85].

More complex data abstractions can be obtained by allowing user-defined data types. Pascal is the only language under consideration that allows user-defined data types [86].

4.2 Control Abstraction

"Control of execution addresses the language features which describe the algorithmic structure of the program" [87]. Without features to describe the algorithmic structure, the modularization process would not be possible. According to Cugini, control of execution includes blocks, subprograms, recursion, and exception handling [88].

4.2.1 Blocks

"The essential features of block structure are a system of program units [(blocks, modules)] that delimit the regions of text and [provide] a method for specifying the names that belong to these regions" [89]. "Blocks allow the programmer to mark off certain sections of code, such that it is treated as a single [statement, and within the block] data may be defined locally which cannot be accessed from outside the block" [90]. The general concept of block structure has already been discussed (See Control Structures, Section 1.2). However, the key concept of local variables and their inaccessibility from outside the block was not stressed.

Now that the identifier scope is understood, the actual block structuring of the languages can be considered. Since identifiers can be declared to be local within any type of block, C is the only language that allows full block structure (See Figure 4.6). By studying Figure 4.6, one can see that C allows the definition of local identifiers within IF-THEN-ELSE constructs (as well as other block structures such as looping). By studying the output, one can see that 1) 'i' is global to the whole program; and 2) 'j' is local to sub1 and is a different entity than the 'j' in sub2, which is local to sub2. Thus, C is the only language under study that has true local identifiers. Due to C's strict type checking, identifiers may not even be referenced (i.e., to print) outside of their defined scope.

Microsoft FORTRAN is not considered a full block structured language [91]. The COMMON statement allows the defining of global variables between subroutines instead of forcing the programmer to pass parameters by reference [92] (See Passing Values, Section 4.2.2.1.1). The COMMON statement passes the memory location of the values [93]. Thus, the corresponding identifiers between COMMON statements must agree in size and type, but may be different names. By studying Figure 4.7, one can see that the variables included in the similarly named COMMON statement (e.g., /BLOCK1/ and /BLOCK2/) are accessible by any subprogram that contains that named COMMON statement (e.g., W and X are accessible by MAIN and by SUB1; Y and Z are accessible by MAIN, SUB1, and SUB2). Data introduced inside a block within a subprogram is global to the whole unit [94] (i.e., variables are accessible outside a block). Data may not be local to blocks alone (i.e., IF-THEN-ELSE constructs, looping constructs, etc.) [95]. By studying Figure 4.8, one can see that a variable (K) introduced inside an IF-THEN-ELSE block can be accessed outside that block. Since FORTRAN has less stringent type checking, FORTRAN will allow referencing variables (i.e., to print) outside of their scope.

QuickBASIC is not considered a full block structured language for the same reasons [96]. QuickBASIC uses the SHARED statement [97] as opposed to the COMMON statement. QuickBASIC does not allow the definition of local variables within IF-THEN-ELSE constructs, looping constructs, and other similar blocks. Standard BASIC is not considered block structured since all variables are considered global through the entire program [98].

Turbo Pascal is not considered a full block structured language since it does not allow the definition of local data in begin-end blocks. While local data can be introduced to procedures, it cannot be introduced inside looping constructs, IF-THEN-ELSE constructs, etc. [99,100]. Pascal will not allow the declaration of such identifiers to occur. By studying the output in Figure 4.9, one can see variable scoping in Pascal. Note that on Sub1 and Sub2 entry, some of the variables (sub1: k,l,m and sub2: n) have been defined, but are not initialized with values. Since sub2 is nested in sub1, sub2 has access to sub1's data. Due to Pascal's strict type checking, variables may not even be referenced (i.e., to print) outside of their defined scope.

4.2.2 Subprograms

Subprograms are considered independent program units that are invoked from a calling program and that operate on arguments passed to it from the calling program (and/or on parameters input from another source) to return the results (if any) to the calling program.

> Subprograms allow the programmer to package computations and parameterize their behavior. There are two [common] forms of subprograms, procedures and functions. A procedure subprogram is a sequence of actions that is invoked as though [the procedure subprogram] were a single statement. A function subprogram is a sequence of computations that result in a single value and is invoked from within an expression. Usually, control returns to the point of invocation after execution of the subprogram, thus forming another one-in, one-out control structure [101].

At this point, program translation must be discussed. Many higher level languages are translated into an equivalent machine-language before being executed. This translation (i.e. "a rendering from one language into another" [102] without changing the meaning [103]) is usually performed in a series of steps.

> Subprograms might first be translated into assembly code [compiler]; assembly code is translated into relocatable machine code [assembler]; units of relocatable code are linked together into a single relocatable unit [linker]; finally, the entire program is loaded into main memory as executable machine code [loader] [104].

In some languages it is possible to separately compile a subprogram, and later link it to the main program [105]. This version of a subprogram is called an external subprogram [106]. Internal (or nested [107]) subprograms are subprograms that are part of the same source file as the invoker [108].

Some languages allow the subprogram to exist separately as a source file (i.e., not compiled). A program may be split into subprograms to be put together again at compile time [109] (e.g., an 'INCLUDE file' command). Thus, at compile time the subprogram is copied to the point in the program where the a reference was made to it and then both the program and subprogram are compiled together. It is important to note that the first subprogram is not compiled until the second program is. This process eases the tedium of constantly retyping subprograms that already exist elsewhere [110]. These subprograms typically exist in libraries [111] (files that hold a collection of these subprograms). These subprograms will be referred to as included subprograms.

4.2.2.1 Procedures

Procedures allow the programmer to write clear and modular programs [112]. The programmer can take frequently executed code and group it together in a procedure. This procedure can then be invoked by a simple call statement as opposed to retyping the whole section of code. Procedures are essential for program readability [113] and for structured programming.

> When procedures are used effectively, they allow the program to be presented in levels of abstractions. The topmost level, the main program, defines the outer structure of the program. The successive lower levels give increasing details about the computations needed to obtain the desired result [114].

Procedures should follow a one-in and one-out philosophy [115].

C defines all of its subprograms as 'functions'. However, the term 'function' has a distinct meaning in this thesis (See Section 4.2.2.2). C's functions are analogous to procedures [116]. Turbo Pascal [117,118] and C [119,120] use internal and included procedures only (See Figures 4.10, 4.11, and 4.12). External procedures may not be used. FORTRAN allows both internal and external [121] procedures (commonly called subroutines) (See Figure 4.13) [122]. QuickBASIC, like FORTRAN, allows both internal and external procedures [123] (QuickBASIC refers to them as subprograms [124]; BASIC does not allow external procedures). QuickBASIC [125] and Microsoft FORTRAN [126] allow the use of included files. QuickBASIC, as well as standard BASIC, allows a simpler form of procedures via the GOSUB statement (QuickBASIC and BASIC refer to them as subroutines [127]. In QuickBASIC and BASIC subroutines, all identifiers are global).

Some languages act as if the actual procedure was inserted at the calling point [128]. The procedure acts "...directly upon the variables that can be referenced at the point where the procedure is invoked" [129]. Thus, variables are not local to the procedure alone. Inadvertent changes to variables that can be referenced at the calling point is possible.

Other languages insist that procedures process all variables as local and "...the effect of the procedure takes place only by assignment to arguments of the call" [130]. This type of a procedure does not have access to the variables that can be referenced at the calling point. Thus, inadvertent changes to these variables is not possible.

In most languages, procedures can use both methods. Thus, "...a procedure may affect its arguments or may affect variables that can be referenced at the point where the procedure is called" [131].

Procedures are not required to exist in total isolation. They are able to communicate with data of the program which invoked the subprogram. Parameters are used as a medium for this communication.

4.2.2.1.1 Passing Values

Subprograms may need parameters to operate on. "...It is through parameters that we generalize the action of a procedure" [132]. It is the effectiveness of parameter passing that determines the usefulness of a procedure. There are two methods that are usually used for data communication between the calling program and the subprogram: 1) passing of data through parameters in the subprogram; and 2) through non-local variables. Non-local variables have already been discussed (See Section 4.1.2) (Standard BASIC uses this method alone).

Values can be passed to subprograms via a parameter list (argument list). "Parameters provide a substitution mechanism which allows the logic of the subprogram to be used with different initial values, thus producing different results" [133].

```
      ...
      CALL SUB1(X,Y,Z)
C X,Y,and Z are in a parameter list.
C  These are the 'actual parameters'
      ...
      END
      SUBROUTINE SUB1(U,V,W)
C U,V, and W are also in a parameter
C  list. These are 'formal parameters'
      ...
```

The subprogram is called via some form of a CALL statement using a list of parameters (arguments), referred to as 'actual parameters' in FORTRAN [134,135]. The actual parameters are passed to the 'formal parameters' (dummy arguments) through the subprogram parameter list [136]. There are several forms of parameter passing. The following sections will discuss these various forms.

4.2.2.1.1.1 Pass by Value

The formal parameter acts as a local variable to the subprogram [137,138] and is initialized with the value associated with the actual parameter [139]. Any changes made to the formal parameter will not effect the actual parameter [140,141] (See Figure 4.14). Therefore, the actual parameter may be any expression, or variable, that has the same type as the formal parameter [142,143]. Parameters passed by value are called 'value parameters' [144]. Pass by value is one method used by Turbo Pascal [145,146] (See Figure 4.15), and QuickBASIC [147] (See Figure 4.16) (not Standard BASIC).

C can only pass parameters by value [148]. "the main distinction is that in C the called [procedure] cannot alter a variable in the calling [procedure]; it can only alter its private, temporary copy" [149]. While this ensures that all variables are local unless specifically arranged otherwise, it does require considerably more effort (than the other languages under consideration) to pass parameters by reference.

> When necessary, it is possible to arrange for a function to modify a variable in the calling routine. The caller must provide the address of the variable to be set (technically a pointer to the variable), and the called function must declare the argument to be a pointer and reference the actual variable indirectly through it [150].

Pointers are discussed in more detail in Chapter 5.

4.2.2.1.1.2 Pass by Reference

The actual parameter must be a variable [151] since the value associated with that variable may change [152]. The formal parameter is local to the subprogram [153]. However, the value associated with the formal parameter is the memory location associated with the actual parameter [154,155]. Thus, changes affecting the formal parameter will effect the actual parameter [156,157,158] (See Figure 4.17). Parameters passed by reference (pass by location [159]) are called 'variable parameters' [160]. Pass by reference is used by FORTRAN [161,162], Pascal [163,164], and QuickBASIC [165].

4.2.2.1.1.3 Pass by Result

The formal parameter is local to the subprogram [166]. Moreover, it must be initialized in the subprogram [167]. After execution of the subprogram, the final value associated with the formal parameter (i.e., the result) is assigned to the actual parameter [168] (See Figure 4.18).

The actual parameter must be a variable [169]. Since pass by result is a rare method of passing, none of the languages under consideration directly support pass by result [170]. However, QuickBASIC, Microsoft FORTRAN, and Turbo Pascal use pass by result to return the values of functions.

4.2.2.1.1.4. Pass by Value-Result

Passing by Value-Result combines the effects of pass by value and pass by result [171]. The formal parameter is local to the subprogram [172]. It is initialized to the value of the actual parameter [173,174]. Upon completion of the subprogram, the final value of the formal parameter is assigned to the actual parameter [175,176]. None of the languages under consideration use pass by value-result.

4.2.2.2 Functions

A function, or "value-returning procedure[,] is one that is invoked as a function reference in an expression" [177]. Value-returning procedures are procedures that act on input data to return a single value [178]. "Value-returning procedures are also known as function procedures or function subprograms" [179].

Value-returning procedures may be used as part of an expression as if it were a variable or a constant [180]. Value-returning procedures (functions) may be intrinsic or user-defined. Intrinsic functions are supplied by a library that accompanies the run time environment [181].

User defined functions can be external or internal. Internal user defined functions are procedures in the same file. In addition, some internal user defined functions may be statement functions.

A statement function is similar to a value-returning procedure, but it is not a procedure [182]. It is a single statement within a program or subprogram [183]. Statement functions may not be referenced outside the program unit in which they are defined [184].

All of the languages under consideration except for QuickBASIC allow the use of value-returning procedures. All of these languages also use intrinsic functions supplied by the compiler and libraries.

Turbo Pascal [185] (See Figure 4.10) and C [186] (See Figure 4.11) use value-returning procedures. They do not have any form of the statement function. Turbo Pascal [187] and C [188] allow internal and external functions.

Microsoft FORTRAN allows both statement functions [189] and function subprograms (See Figure 4.13) [190]. FORTRAN's functions may be internal or external.

Standard BASIC allows internal functions by use of the GOSUB statement. Standard BASIC does not allow any form of the statement function [191]. QuickBASIC does not have function subprograms and does not allow statement functions to be in subprograms [192]. QuickBASIC allows a multi-line statement function (i.e., a series of statements grouped together in a block that is defined as a form of a statement function [193]) only in the main program [194]. However, statement functions in QuickBASIC are global to the entire compilation unit [195,196]. Thus, a statement function declared in the main program can be referenced in a subprogram as long as the subprogram is compiled with the program that defines the function. It is also possible to emulate a function using another subprogram. An additional subprogram must be called where the passed parameters must include an identifier for the desired value and an assignment must be made to that identifier in the called subprogram.

4.2.3 Recursion

Recursion is the ability of a procedure or subprogram to call itself [197]. Recursion offers solutions to problem solving that are not available in many languages. Recursion and iteration are occasionally confused. It is important to distinguish the two.

> Iteration is the repeated performance of same action until a particular condition is met. The performance is carried to completion each time before the testing of the condition that decides if the performance is to be repeated [198].

> Recursion involves self-nesting. The performance is not carried to completion before the condition is tested. Instead, the condition is examined as part of the performance of the action and, if the condition is not satisfied, the action is performed again as a subroutine of the as yet uncompleted original action [199].

There is also a difference in the data references. Iteration occurs in a single environment. Each time the routine is invoked, the same variables are affected. In recursion, the routine is executed in a new environment each time it is invoked. Thus, a new set of variables is created each time the routine is called.

Everything that is done recursively can be done iteratively [200]. "In theory, any recursive program can be converted into an iterative program..." [201].

> This is effectively what [a] ...Pascal compiler does; it converts recursion into iteration. Although this reduction is a theoretical possibility, it is not practical to do by hand in most cases, since the resulting program is so much more complicated. From the programmers point of view, recursion is more powerful than iteration [202].

However, everything that is done iteratively may not be done recursively. An iterative solution requires that storage be set aside in advance. By using recursion, storage is automatically increased or decreased as execution progresses [203] (since storage allocation is performed each time the subprogram is called).

A subprogram is directly recursive if it calls itself [204]. A subprogram "...is indirectly recursive if it calls another subprogram that calls the original subprogram or that initiates a further call chain of subprogram calls that eventually leads back to a call of the original subprogram" [205].

"Recursion, in the form of recursive subprogram calls, is one of the most important sequence control structures in programming. Many algorithms are most naturally represented using recursion" [206]. However, "Recursive calling structures are seldom supported directly by the hardware and thus must be largely software-simulated, leading to a major difference in run-time efficiency between the nonrecursive and recursive mechanisms" [207].

The process of storing the order of recursive calls is usually done via a stack. On the first (nonrecursive) call to the subprogram, the return point is stored in the first stack location. On the first recursive call the return point is stored in the second stack location. On the second recursive call, the return point is stored in the third stack location, and so on. Thus, when the subprogram is completed, the last stack entry contains the return point. This value is used and deleted off of the stack to reveal the next return point, which is used and deleted, and so on. Thus "...the stack of return locations allows the subprogram to unwind the nest of recursive calls correctly, returning at the end to the original calling program" [208]. This general process can be used to store the return points for all subprogram calls (i.e., those that are and are not recursive) as well.

Only C [209] and Pascal [210] allow recursion. For a brief example of recursion, see figure 4.19.

4.2.4 Exception Handling

"There are 'exception' conditions that can arise in every program. Input data may contain values that are out of range, a hardware unit may fail,..." [211] and a variety of other unforeseen circumstances.

Exception conditions should not be thought of as purely errors. Some exceptions are necessary to the normal operations of the program (e.g., end of file) [212].

> For our purposes here, we define an exception condition as: A condition that prevents the completion of the operation that detects it, that cannot be resolved within the local context of the operation, and that must be brought to the attention of the operation's invoker. The process of bringing the condition to the invoker's attention is called raising the exception. The corresponding action by the invoker is called handling the exception [213].

It is considered a good programming practice to be able to handle these exceptions within the program so that the program will terminate under normal conditions [214].

There are two general classes of exception handling. A domain failure occurs when "the input parameters to the operation do not satisfy the requirements of the operation" [215] (e.g., filling an array and exceeding the upper bound of a subscript [216], inputting a character when a integer is expected, etc.). A range failure occurs when "the operation is unable to produce a result that is in its range" [217] (e.g., an overflow error [218]).

The programmer should be able to dictate how an exception will be handled [219]. This response could be simply printing out the value of some variables or actual procedures to determine the cause of the exception [220]. "The basic operation of an exception handler is to perform some diagnostic or repair actions" [221] [222].

"With most methods of exception handling, the flow of control passes to some remote program text that defines the action to be taken when the exception is raised. Thus, the handler may be viewed as a sort of *trap*" [223]. There are two general views as to whether or not the program should resume normal execution after the exception has been handled.

One view states that exception handlers are basically subprograms that assume control once the exception is raised [224]. Maintaining that the handler is a subprogram, control should resume at the statement following the exception [225]. "With this view, the idea of a trap is still retained, but resumption of normal program flow is implicit" [226].

Another view is that exceptions are program errors. As such, normal program execution should not continue. "The primary role of an exception handler is to provide some appropriate clean-up operation before termination" [227]. Thus, normal program execution would not continue, and a clean termination would occur.

If the programmer does not handle the exception, the program is in error and it will terminate [228]. "The conventional response to an error situation in a programming language is a simple abnormal termination of a program, usually with the printing of some diagnostic message" [229].

QuickBASIC allows exception handling with the ON ERROR GOTO statement [230]. Microsoft FORTRAN has minimal error handling with it's READ, WRITE, or INQUIRE statements [231] (Standard FORTRAN has no exception handling). Pascal and C have no forms of exception handling.

4.3 Conclusions

Semantic structures are features that allow the programmer to represent the meaning of an algorithm or data or both. It is through the semantic structure of a language that a programmer is able to use structured design and structured programming to decompose a problem into coded modules. Without good semantic structure, a programmer would find it difficult to use the control structures to form modules so that the structured design process could be completed.

A stack based language is preferable to a static language since memory allocation is more efficient. FORTRAN and QuickBASIC are static languages. C and Pascal allow more versatility by being stack based languages with some dynamic implementations. Thus, FORTRAN restricts the programmer to reserve memory prior to execution, and thereby eliminates the use of arrays that vary in size.

Structured programming advocates the use of local variables and discourages the use of non-local variables. The use of local variables significantly reduces the chance of side effects, enhances the modularity of a subprogram, increases the efficiency of storage use, and increases the readability of a program. The use of global variables increases the responsibility of a programmer by requiring him/her to selectively choose variable names so that changes made to a variable do not effect a variable of the same name elsewhere in the program. Microsoft C is the only language under consideration that fully supports local data. Microsoft FORTRAN, Microsoft QuickBASIC, and Turbo Pascal allow local data only to subprograms.

Structured programming considers external data to be a bad programming practice. C, FORTRAN, and QuickBASIC allow external variables. However, FORTRAN and QuickBASIC force the programmer to use declarations in each subprogram that uses these variables.

Data abstraction can aid in subsequent program modification by allowing variable type to be assigned to identifiers. C and Pascal are the only languages that allow data abstraction of any form. Pascal allows higher and more complex forms of data abstraction.

A language's support of block structure can enable or disable the use of structured programming building blocks. Block structure is intended to localize data, to aid in modularizing a program, and most importantly, to enforce the one control/logic entrance and one control/logic exit concept. C is the only language under consideration that fully utilizes block structure. After C, Pascal is considered more block structured than FORTRAN, QuickBASIC, and BASIC (in decreasing order of block structure facilities). FORTRAN and QuickBASIC are approximately equal in terms of block structuring. BASIC is lacking in this area. A programmer should utilize block structuring to its full advantages.

Subprograms aid in maintaining the modularity and block structure of a program. C, Pascal, FORTRAN, and QuickBASIC all support subprograms with local or global variables (BASIC does not support local variables). The use of local variables in subprograms is strongly encouraged.

All the languages under consideration allow internal and included subprograms. C and Turbo Pascal do not allow external subprograms. External subprograms allow the quick development of modular programs since they can be referenced without having to recompile them. However, they have a tendency to decrease readability since the source code is not embedded in the program itself. As long as the source code is available, included subprograms and external subprograms actually accomplish the same result. In either case, a subprogram that exists in a separate file can be accessed. The only important difference is that the external subprogram is compiled and the included subprogram is not.

Engineering applications make extensive use of functions. QuickBASIC is the only language that does not support function subprograms. Worse still, it will not allow statement functions of any form in its subprograms (although when declared in the main program, they are global to the whole compilation unit). FORTRAN supports statement functions that may appear in any program unit. While Turbo Pascal and C do not utilize statement functions, functions (and statement functions) may be implemented using value-returning procedures.

A language should support as many types of parameter passing as is possible since it is necessary to use the different types at different times. Pass by reference is important since it allows changes to be made to the actual parameter. Pass by value is the only form of parameter passing that should be used with functions. Turbo Pascal and QuickBASIC allow both passing by value and by reference. While C directly supports only value parameters, reference parameters can be emulated using pointers. FORTRAN only supports pass by reference. However, in order to maintain the security of a program, variables should be local to procedures whenever possible. Thus, pass by value is preferable to pass by reference.

Turbo Pascal and C are the only languages under consideration that allow unique solutions using recursion. A language should support recursion since it is the only technique available for certain solutions.

Exception handling is the key to allowing a programmer to anticipate problems and prepare for them. In this manner the programmer can force a normal termination of the program as opposed to abnormal termination if the exception was not handled. Exception handling should be utilized by user-friendly programs. Microsoft FORTRAN has minimal error handling on I/O operations. QuickBASIC allows a broader range of exception handling.

```
...
  -- variable B cannot be referenced
          outside procedure P nor can
          procedure C be invoked from
          outside procedure P.
...
P: procedure
   B: integer;
   C: procedure
      ...
   end;
begin
  --statements for P
end;
...
```

Figure adapted from: Marcotty, Michael and Henry F. Ledgard Programming Language Landscape 2nd ed. (Chicago, Illinois: Science Research Associates, Inc., 1986) p.337.

FIGURE 4.1 (A) - LOCAL SCOPE

```
...
x: integer;      --outer x
A: procedure
   x: integer;      --inner x
   B:procedure
      ...
   begin
      ...      --the outer x cannot be referenced here
   end;
begin
...
end;
```

Figure adapted from: Marcotty, Michael and Henry F. Ledgard Programming Language Landscape 2nd ed. (Chicago, Illinois: Science Research Associates, Inc., 1986) p.337.

FIGURE 4.1 (B) - LOCAL VARIABLE

```
program
   A,B,C : integer;               -- variables A,B,C

   Q: procedure
   begin                          -- line 1
      A:=A+2;                     -- line 2
      C:=C+2;                     -- line 3
   end;

   R: procedure
      C: integer;                 -- variable R.C
   begin                          -- line 4
      C:=2;                       -- line 5
      Q;                          -- line 6
      B:=A+B;                     -- line 7
      output A,B,C;               -- line 8
   end;

   S: procedure
      B,C: integer                -- variable S.B and S.C
      Q: procedure
      begin                       -- line 9
         A:=A+1;                  -- line 10
         C:=C+1;                  -- line 11
      end;
   begin                          -- line 12
      B:=3;                       -- line 13
      C:=1;                       -- line 14
      Q;                          -- line 15
      R;                          -- line 16
   end;

begin                             -- line 17
   A:=1;                          -- line 18
   B:=1;                          -- line 19
   C:=1;                          -- line 20
   R;                             -- line 21
   S;                             -- line 22
end;
```

Figure taken from: Marcotty, Michael and Henry F. Ledgard Programming Language Landscape 2nd ed. (Chicago, Illinois: Science Research Associates, Inc., 1986) p.333.

FIGURE 4.2 - LOCAL VARIABLE

line	A	B	S.B	C	R.C	S.C	Notes
17	?	?	-	?	-	-	The outermost three local variables are undefined
18	1	?	-	?	-	-	A is assigned 1
19	1	1	-	?	-	-	B is assigned 1
20	1	1	-	1	-	-	C is assigned 1
21	1	1	-	1	-	-	Procedure R is invoked
4	1	1	-	1	?	-	R's local C is created
5	1	1	-	1	2	-	R's C is assigned 2
6	1	1	-	1	2	-	Procedure Q is invoked
1	1	1	-	1	2	-	
2	3	1	-	1	2	-	A is incremented by 2
3	3	1	-	3	2	-	C is incremented by 2, Q is completed
7	3	4	-	3	2	-	
8	3	4	-	3	2	-	A=3,B=4,C=2 is printed; R is completed; R's C ceases to exist; Control returns to main

FIGURE 4.3 - EXECUTION OF FIG. 4.2

22	3	4	-	3	-	-	Procedure S is invoked
12	3	4	?	3	-	?	S's B and C are created
13	3	4	3	3	-	?	S's B is assigned 3
14	3	4	3	3	-	1	S's C is assigned 1
15	3	4	3	3	-	1	Procedure Q is invoked
9	3	4	3	3	-	1	
10	4	4	3	3	-	1	A is incremented
11	4	4	3	3	-	2	S's C is incremented; Q is completed
16	4	4	3	3	-	2	Procedure R is invoked
4	4	4	3	3	?	2	R's local C is created
5	4	4	3	3	2	2	R's C is assigned 2
6	4	4	3	3	2	2	Procedure Q is invoked
1	4	4	3	3	2	2	
2	6	4	3	3	2	2	A is incremented by 2
3	6	4	3	5	2	2	C is incremented by 2, Q is completed
7	6	10	3	5	2	2	
8	6	10	3	5	2	2	A=6,B=10,C=2 is printed; R is completed; Control returns to S; S is completed; control returns to main, which is completed.

Figure adapted from: Marcotty, Michael and Henry F. Ledgard Programming Language Landscape 2nd ed. (Chicago, Illinois: Science Research Associates, Inc., 1986) p.333.

FIGURE 4.3 (CONT) - EXECUTION OF FIG. 4.2

Program Listing 1:

```
100 INPUT "TIME = ";T
110 GOSUB 500
120 PRINT "TIME T = ";T
130 END
500 REM SUBROUTINE THERMO
510 INPUT "TEMPERATURE = ";T
520 IF T > 75 THEN 530 ELSE 550
530     PRINT "AIR CONDITIONER ON"
540     GOTO 570
550   REM ELSE
560     PRINT "AIR CONDITIONER OFF"
570   REM ENDIF
580 RETURN
```

Program Listing 2:

```
100 INPUT "TIME = ";T
110 GOSUB 500
120 PRINT "TIME T = ";T
130 END
500 REM SUBROUTINE THERMO
510 INPUT "TEMPERATURE = ";THT
520 IF THT > 75 THEN 530 ELSE 550
530     PRINT "AIR CONDITIONER ON"
540     GOTO 570
550   REM ELSE
560     PRINT "AIR CONDITIONER OFF"
570   REM ENDIF
580 RETURN
```

Output:

```
   Program 1                      Program 2

TIME = ? 1200                 TIME = ? 1200
TEMPERATURE = ? 83            TEMPERATURE = ? 83
AIR CONDITIONER ON            AIR CONDITIONER ON
TIME = 83                     TIME = 1200
```

Figure adapted from: Chapra, Steven C. and Raymond P. Canale Introduction to Computing for Engineers (New York, New York:: McGraw-Hill, Inc. 1986) p.195.

FIGURE 4.4 - GLOBAL VARIABLE

Program Listing:

```
program
   A: integer;                      -- Global variable A

   PURELY_GLOBAL: procedure
   begin
      A:=5;
      output A;
   end;

begin
   A:=1;
   output A;
   PURELY_GLOBAL;
   output A;
end;
```

Output:

```
A=1
A=5
A=5
```

Figure adapted from: Marcotty, Michael and Henry F. Ledgard Programming Language Landscape 2nd ed. (Chicago, Illinois: Science Research Associates, Inc., 1986) p.332.

FIGURE 4.5 - GLOBAL VARIABLE

Program Listing:

```
#define limit 10
int i;                      /* i is global */
sub2()
{
int j;                      /* j is local to sub2 and   */
   j=2;                     /*    is not related to the */
   i=i+5;                   /*    j in sub1             */
   printf("sub2 j,i                   %d   %d\n",j,i);
}
sub1()
{
int j;                      /* j is local to sub1 and   */
   j=2;                     /*    is not related to the */
   if (i=1)                 /*    j in sub2             */
     {
      int l;                    /* l is local to the */
      for (l=1;l<=limit;++l)    /*    if block only   */
       { ++j;++i;
         printf("%d  ",l);}
     }
   else
    { ++i; }
   printf("\nsub1 before sub2 j,i     %d  %d\n",j,i);
   sub2();
   printf("sub1 after sub2 j,i      %d  %d\n",j,i);
}
main()
{
 i=1;
 sub1();
 ++i;
 printf("main i                             %d\n",i);
}
```

```
Output:

1  2  3  4  5  6  7  8  9  10
sub1 before sub2 j,i     12  11
sub2 j,i                 2   16
sub1 after sub2 j,i      12  16
main i                       17
```

FIGURE 4.6 - C BLOCKS

Program Listing:
```
      COMMON /BLOCK1/ W,X
      COMMON /BLOCK2/ Y,Z
      W=1
      X=1
      Y=1
      Z=1
      N=1
        WRITE (6,'(A,A)')  '                          ',
     $                    ' M  N  K  W   X   Y   Z'
        WRITE (6,10) 'MAIN BEFORE SUB1 =',M,N,K,W,X,Y,Z
      CALL SUB1(N)
        WRITE (6,10) 'MAIN AFTER  SUB1 =',M,N,K,W,X,Y,Z
   10 FORMAT (1X,A,3(2X,I1),4(2X,F2.0))
      END
      SUBROUTINE SUB1(M)
      COMMON /BLOCK1/ W,X
      COMMON /BLOCK2/ Y,Z
        WRITE (6,10) 'SUB1 ON ENTRY    =',M,N,K,W,X,Y,Z
      M=M+2
      W=W+2
      K=K+2
        WRITE (6,10) 'SUB1 BEFORE SUB2 =',M,N,K,W,X,Y,Z
      CALL SUB2()
        WRITE (6,10) 'SUB1 AFTER SUB2  =',M,N,K,W,X,Y,Z
   10 FORMAT (1X,A,3(2X,I1),4(2X,F2.0))
      END
      SUBROUTINE SUB2()
      COMMON /BLOCK2/ Y,Z
       WRITE (6,10) 'SUB2 ON ENTRY    =',M,N,K,W,X,Y,Z
      M=M+3
      Z=Z+3
      K=K+3
       WRITE (6,10) 'SUB2 ON EXIT     =',M,N,K,W,X,Y,Z
   10 FORMAT (1X,A,3(2X,I1),4(2X,F2.0))
      END
```
Output:
```
                    M  N  K  W   X   Y   Z
 MAIN BEFORE SUB1 = 0  1  0  1.  1.  1.  1.
 SUB1 ON ENTRY    = 1  0  0  1.  1.  1.  1.
 SUB1 BEFORE SUB2 = 3  0  2  3.  1.  1.  1.
 SUB2 ON ENTRY    = 0  0  0  0.  0.  1.  1.
 SUB2 ON EXIT     = 3  0  3  0.  0.  1.  4.
 SUB1 AFTER SUB2  = 3  0  2  3.  1.  1.  4.
 MAIN AFTER  SUB1 = 0  3  0  3.  1.  1.  4.
```

FIGURE 4.7 - FORTRAN SCOPE

Program Listing:

```
      INTEGER I,J,N
      REAL P
      I=1
      J=1
      IF (J.EQ.1) THEN
         K=K+1
      ELSE
         K=K+2
      ENDIF
      K=K+2
      N=1
      P=1
       WRITE (6,'(A,A)') '                ',
     $                   '  I    J    N   P    O     M    K'
       WRITE (6,10) 'BEFORE SUB1',I,J,N,P,O,M,K
      CALL SUB1(N,P)
       WRITE (6,10) 'AFTER SUB1 ',I,J,N,P,O,M,K
 10   FORMAT (1X,A,3(2X,I2),2(2X,F2.0),2(2X,I2))
      END

      SUBROUTINE SUB1(M,O)
      INTEGER I,J,M
      REAL O
       WRITE (6,10) 'ENTRY SUB1 ',I,J,N,P,O,M,K
      I=I+2
      J=J+2
      M=I
      O=J
       WRITE (6,10) 'EXIT  SUB1 ',I,J,N,P,O,M,K
 10   FORMAT (1X,A,3(2X,I2),2(2X,F2.0),2(2X,I2))
      END
```

Output:

```
                 I   J   N   P    O    M   K
 BEFORE SUB1     1   1   1   1.   0.   0   3
 ENTRY SUB1      0   0   0   0.   1.   1   0
 EXIT  SUB1      2   2   0   0.   2.   2   0
 AFTER SUB1      1   1   2   2.   0.   0   3
```

FIGURE 4.8 - PRACTICAL FORTRAN BLOCKING

Program Listing:
```
program sample;
var
  i   :integer;
  procedure subl;
     var
        k,l : integer;
        m   : real;
        procedure sub2;
        var
             n :integer;
        begin
            writeln('sub2  entry       =',i:5,k:5,l:5,
                     m:5, n:5);
            i:=i+2;
            n:=l+i+k;
            k:=k+l;
            writeln('sub2  exit        =',i:5,k:5,l:5,
                    m:5,n:5);
        end; (* sub2 *)
  begin (* subl *)
     writeln('subl  entry       =',i:5,k:5,l:5,m:5);
     i:=i+1;
     k:=2;
     l:=1;
     m:=1.0;
     writeln('subl before sub2 =',i:5,k:5,l:5,m:5);
     sub2;
     writeln('subl after  sub2 =',i:5,k:5,l:5,m:5);
     end;    (* subl *)
begin  (* main starts here *)
   i:=1;
   writeln('main before subl =',i:5);
   subl;
   writeln('main after  subl =',i:5);
end.   (* main *)
```

Output:
```
main before sub   =    1
subl entry        =    1 -248-64141.61E+34
subl before sub2 =    2    2    1   1.0E+0C
sub2 entry        =    2    2    1   1.0E+00   -6416
sub2 exit         =    4    3    1   1.0E+00       7
subl after sub2  =    4    3    1   1.0E+00
main after subl  =    4
```

FIGURE 4.9 - PASCAL BLOCKS

```
PROGRAM MAIN(INPUT,OUTPUT);   (*  Test Power Function *)
  VAR
     I     : INTEGER;

  FUNCTION POWER(X,N:INTEGER):REAL;

(* Raise x to n-th power; n>0 *)
     VAR
       P : REAL;
       K : INTEGER;

     BEGIN
         K:=0;
         P:=1;
         REPEAT
             K:=K+1;
             P:=P*X
         UNTIL K=N;
      POWER:=P;
   END; (* Function Power *)

   PROCEDURE INCOUT(var i : integer);
    BEGIN
     WRITELN (I:2,POWER(2,I):8:2,POWER(-3,I):10:2)
   END; (* incout *)

BEGIN
  I:=0;
  REPEAT
     I:=I+1;
     INCOUT(I)
  UNTIL I=10;
END. (* Program main *)
```

Example adapted from: Kernighan, Brian W. and Dennis M. Ritchie The C Programming Language (Englewood Cliffs, New Jersey: Prentice Hall Inc., 1978),pp.22-23.

FIGURE 4.10 - PASCAL SUBROUTINE & FUNCTION

```
/* this demonstrates use of functions
   and passing parameters by reference

   function to return user input value */
int getvalue()
{ int inputvalue;
 /*get an integer value from user at standard input*/
 printf ("Input a value:");
 scanf ("%d",&inputvalue);
 printf ("Inputvalue: %d\n",inputvalue);
 /*return value for function name*/
 return(inputvalue);
}/*end function getvalue*/

/*function to swap values*/
int swap (firstnumber, secnumber)
int *firstnumber,*secnumber;
{
 int tempstorage;
 printf ("Original order:%d %d\n", *firstnumber,
         *secnumber);
 tempstorage=*firstnumber;
 *firstnumber=*secnumber;
 *secnumber=tempstorage;
 printf("swapped order:%d %d\n", *firstnumber,
        *secnumber);
/*return no value for function name*/
return;
}

/*main:get two numbers from user and swap values*/
main()
{ int firstvalue,secvalue;

 firstvalue=getvalue();
 secvalue=getvalue();
 printf("Firstvalue:%d Secvalue: %d\n",firstvalue,
        secvalue);

swap (&firstvalue,&secvalue);
}
```

Example taken from: Brown, Douglas L. From Pascal to C (Belmont, California: Wadsworth Publishing Co., 1985), p.108.

FIGURE 4.11 - PROCEDURES AND FUNCTIONS IN C

```
/* file stdio.h contains all basic input and output
     functions (i.e., printf, getchar, fprintf, etc.).
     In standard library.                    */

#include <stdio.h>
#define MAXLEN 255


/* function ltou checks array of characters,
     outputs character if uppercase,
     converts to uppercase if lowercase
   function ltou(upline;char) is in library.
     Returns int.
     Will be loaded at compile time.          */

#include <ltou.h>

/* function getline gets a line of input from standard
     input character by character, and continues until
     EOF or a carriage return is encountered.
   function getline(dataline[];char,limit;int) is in
     library.  Returns int.
     Will be loaded at compile time           */


/* Main function gets line of characters and calls
    function ltou to process line.  Process will end
    when an empty string is entered at keyboard */


main()
 {
 int maxchar,len;
 char inputline [MAXLEN],line[MAXLEN];
 maxchar=0;
 while ((len=getline(inputline,MAXLEN))>0)
     if ((maxchar=ltou(line)=255)
        len=0;
}
```

Figure adapted from: Brown, Douglas L. From Pascal to C (Belmont, California: Wadsworth Publishing Company, 1985), pp,90-91.

FIGURE 4.12 - INCLUDED FILES IN C

```
C Main Program: Test Power Function
C
      REAL J,POW
      INTEGER I,K
C
C The following statement is a statement function
C
      POW(J,K)=J**K
      DO 20 I=1,10
         CALL OUT(I)
         WRITE (*,10) 'Statement Value:',I,POW(2.0,I)
   10    FORMAT (1X,A,1X,I2,2X,F7.2)
   20 CONTINUE
      STOP
      END

      SUBROUTINE OUT(I)
      REAL POWER
      INTEGER I
C
C This Subroutine can be compiled separately
C
      WRITE (*,30) 'Subroutine Value:',I,POWER(2,I)
   30 FORMAT (1X,A,1X,I2,2X,F7.2)
      RETURN
      END

      REAL FUNCTION POWER(X,N)
      INTEGER X,N
C
C Raise x to n-th power; n>0
C This function can also be compiled
C    separately.
C
       POWER=1.0
       DO 40 I=1,N
   40   POWER=POWER*X
      RETURN
      END
```

Example adapted from: Kernighan, Brian W. and Dennis M. Ritchie The C Programming Language (Englewood Cliffs, New Jersey: Prentice Hall Inc., 1978),pp.22-23.

FIGURE 4.13 - FORTRAN SUBROUTINE & FUNCTION

```
program
  I: integer;
  A: integer array;

  SWAP_BY_VALUE: procedure(X: value, Y: value)
     TEMP: integer;
  begin
     TEMP := X;
     X    := Y;
     Y    := TEMP;
  end;

begin
   I    :=3;
   A[I] :=6;
   output I, A[3];
   SWAP_BY_VALUE(I,A[I]);
   output I, A[3];
end;
```

Output

```
   I = 3    A[3] = 6
   I = 3    A[3] = 6
```

Figure taken from: Marcotty, Michael and Henry F. Ledgard Programming Language Landscape 2nd ed. (Chicago, Illinois: Science Research Associates, Inc., 1986) p.300.

FIGURE 4.14 - PASS BY VALUE

Variable parameters:

```
procedure Switch(var A,B:Integer);
Var tmp:Integer;
begin
   tmp:=A;A:=B;B:=tmp;
end;
```

Value Parameters:

```
Procedure Switch(A,B:integer);
Var tmp:integer;
begin
  tmp:=A;A:=B;B:=tmp;
end;
```

Example taken from: Borland International, Turbo Pascal Reference Manual (Scotts Valley, California: Borland International, Inc., 1986), p.132.

FIGURE 4.15 - PARAMETER PASSING IN PASCAL

Program Listing:

```
A=5 : B=0
PRINT "Initial A=" A " B="B
CALL SQUARE (A,(B))
' Note that A is passed by reference.
'           B is passed by value.
PRINT "Main program variable values are " A "and " B
END

SUB SQUARE(X,Y) STATIC
   Y=X*X
   PRINT "In subprogram, Y= " Y
   X=X/2
END SUB
```

Output:

```
Initial A=5 and B=0
In subprogram Y=25
Main program variable values are 2.5 and 0
```

Example taken from: Microsoft QuickBASIC Reference Manual (Redmond, Washington: Microsoft Corporation, 1986), pp.118-119.

FIGURE 4.16 - PARAMETER PASSING IN QUICKBASIC

```
program
  I: integer;
  A: integer array;

  SWAP_BY_LOCATION: procedure(X: location, Y: location)
     TEMP: integer;
  begin
     TEMP := X;
     X    := Y;
     Y    := TEMP;
  end;

begin
   I    :=3;
   A[I] :=6;
   output I, A[3];
   SWAP_BY_LOCATION(I,A[I]);
   output I, A[3];
end;
```

Output

```
   I = 3    A[3] = 6
   I = 6    A[3] = 3
```

Figure taken from: Marcotty, Michael and Henry F. Ledgard Programming Language Landscape 2nd ed. (Chicago, Illinois: Science Research Associates, Inc., 1986) p.301.

FIGURE 4.17 - PASS BY REFERENCE (LOCATION)

```
program
  I: integer;
  A: integer array;

  SWAP_BY_RESULT: procedure(X: result, Y: result)
     TEMP: integer;
  begin
     TEMP := X;
     X    := Y;
     Y    := TEMP;
  end;

begin
   I    :=3;
   A[I] :=6;
   output I, A[3];
   SWAP_BY_VALUE(I,A[I]);
   output I, A[3];
end;
```

Output

```
 I = 3    A[3] = 6
 *** ERROR: ATTEMPT TO EVALUATE AN UNDEFINED VARIABLE X
```

Figure taken from: Marcotty, Michael and Henry F. Ledgard Programming Language Landscape 2nd ed. (Chicago, Illinois: Science Research Associates, Inc., 1986) p.303.

FIGURE 4.18 - PASS BY RESULT

Program Listing:

```
program Factorial;
  var
    number: integer;


  function Factorial(Value: Integer): Real;
  begin
    if Value=0 then
      Factorial:=1
    else
      Factorial:=Value*Factorial(Value-1);
    end;


begin
  number:=9;
  writeln ('The factorial of ',number:3,' is
',Factorial(number):7:2);
end.
```

Program Output:

```
The factorial of  9 is 362880.00
```

Figure adapted from: Borland, Turbo Pascal (Scotts Valley, California: Borland International, 1986), p.138.

FIGURE 4.19 - RECURSION IN PASCAL

NOTES - CHAPTER 4

1 Webster's New Collegiate Dictionary, p.1044.

2 Cugini, p.7.

3 Ghezzi, Carlo and Mehdi Jazayeri Programming Language Concepts 2nd ed. (New York, New York: John Wiley & Sons, 1987), p.26.

4 Cugini, p.7.

5 Cugini, p.12.

6 Cugini, pp.12-14.

7 Ghezzi, p.52.

8 Ghezzi, p.52.

9 Ghezzi, p.52.

10 Lee, Personal Interview, 4/29/87.

11 Lee, Personal Interview, 4/29/87.

12 Lee, Personal Interview, 4/29/87.

13 Lee, Personal Interview, 4/29/87.

14 Lee, Personal Interview, 4/29/87.

15 Lee, Personal Interview, 4/29/87.

16 Ghezzi, p.52.

17 Ghezzi, p.52.

18 Ghezzi, p.52.

19 Ghezzi, p.52.

20 Ghezzi, p.52.

21 Ghezzi, p.52.

22 Ghezzi, p.59.

23 Ghezzi, p.59.

24 MacLennan, p.106.

25 Marcotty, p.341.

26 Ghezzi, p.60.

27 Ghezzi, p.60.

28 American National Standards Committee X3, p.107.

29 Marcotty, p.340.

30 MacLennan, p.106.

31 MacLennan, p.106.

32 Ghezzi, p.60.

33 Ghezzi, p.60.

34 Ghezzi, p.60.

35 Ghezzi, p.78.

36 Ghezzi, p.78.

37 Ghezzi, p.60.

38 Ghezzi, p.60.

39 Ghezzi, p.74.

40 Ghezzi, p.78.

41 Kernighan, Brian W. and Dennis M. Ritchie, The C Programming Language (Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1978), p.76.

42 Marcotty, p.521.

43 American National Standards Committee X3, p.74.

44 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.29.

45 Barron, pp.66-67.

46 Lee, Personal Interview 2/24/87.

47 American National Standards Committee X3, p.57.

48 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.29.

49 Ghezzi, p.59.

50 Ghezzi, p.53.

51 Marcotty, p.336.

52 Marcotty, p.336.

53 Marcotty, p.337.

54 Marcotty, p.337.

55 Chapra, p.194.

56 Marcotty, p.338.

57 Marcotty, p.338.

58 Chapra, p.194.

59 Chapra, p.194.

60 Cugini, p.14.

61 Ghezzi, p.60.

62 Ghezzi, p.60.

63 Cugini, p.14.

64 Kernighan, p.77.

65 Kernighan, p.28.

66 Kernighan, p.28.

67 Lee, Personal Interview, 4/29/87.

68 Kernighan, p.76-77.

69 Ghezzi, p.60.

70 Marcotty, p.336.

71 Cugini, p.14.

72 Merchant, FORTRAN77 Language, p.386.

73 Microsoft, QuickBASIC reference Manual, p.456.

74 Cugini, p.14.

75 Merchant, FORTRAN77 Language, pp.386-389.

76 Microsoft, QuickBASIC reference Manual, p.456.

77 Cugini, p.14.

78 American National Standards Committee X3, p.13.

79 Ghezzi, p.22.

80 MacLennan, p.67.

81 Cugini, p.14.

82 Cugini, p.14.

83 Cugini, p.14.

84 Cugini, p.14.

85 Kernighan, p.140.

86 Cugini, p.14.

87 Cugini, p.7.

88 Cugini, p.vi-vii.

89 Marcotty, p.336.

90 Cugini, p.9.

91 Cugini, p.9.

92 Lee, Personal Interview 2/9/87.

93 Lee, Personal Interview, 4/29/87.

94 Lee, Personal Interview 2/24/87.

95 Lee, Personal Interview 2/24/87.

96 Microsoft, QuickBASIC reference Manual, p.122.

97 Microsoft, QuickBASIC reference Manual, p.456.

98 Chapra, p.194.

99 Cugini, p.9.

100 Borland, p.319.

101 Marcotty, p.293.

102 Webster's New Collegiate Dictionary, p.1232.

103 Lee, Personal Interview, 4/29/87.

104 Ghezzi, p.51.

105 Microsoft, Microsoft FORTRAN Compiler User's Guide (Redmond, Washington: Microsoft Corporation, 1985), p.56.

106 Cugini, p.9.

107 Cugini, p.9.

108 Cugini, p.9.

109 Borland, p.147.

110 Borland, p.148.

111 Borland, p.147.

112 Marcotty, p.303.

113 Marcotty, p.303.

114 Marcotty, p.303.

115 Marcotty, p.304.

116 Kernighan, p.22.

117 Borland, p.147-148.

118 Cugini, p.9.

119 Kernighan, p.23.

120 Kernighan. p.23.

121 Merchant, FORTRAN77 Language, pp.254-255.

122 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.4.

123 Microsoft, QuickBASIC reference Manual, p.133.

124 Microsoft, QuickBASIC reference Manual, p.576.

125 Microsoft, QuickBASIC reference Manual, p.188.

126 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.185.

127 Microsoft, QuickBASIC reference Manual, p.577.

128 Marcotty, p.304.

129 Marcotty, p.304.

130 Marcotty, p.304.

131 Marcotty, p.304.

132 Marcotty, p.306.

133 Borland, p.127.

134 Marcotty, p.294.

135 Lee, Personal Interview 4/21/87.

136 Marcotty, p.294.

137 Marcotty, p.297.

138 Borland, p.127-129.

139 Marcotty, p.297.

140 Marcotty, p.297.

141 Borland, p.127-129.

142 Marcotty, p.297.

143 Borland, p.127-129.

144 Borland, p.127-129.

145 Marcotty, p.307.

146 Borland, pp.127.

147 Microsoft, QuickBASIC reference Manual, pp.118

148 Brown, Douglas L. From Pascal to C (Belmont, California: Wadsworth Publishing Company, 1985), pp.105-107.

149 Kernighan, p.24.

150 Kernighan, p.24.

151 Marcotty, p.298.

152 Foutz, Personal Interview, 4/29/87.

153 Marcotty, p.298.

154 MacLennan, p.54.

155 Marcotty, p.298.

156 Marcotty, p.298.

157 Borland, p.128.

158 MacLennan, p.54.

159 Marcotty, p.298.

160 Borland, p.128.

161 Marcotty, p.309.

162 Microsoft, Microsoft FORTRAN Compiler Reference Manual, pp.60-61.

163 Marcotty, p.309.

164 Borland, pp.128.

165 Microsoft, QuickBASIC reference Manual, pp.118.

166 Marcotty, p.297.

167 Marcotty, p.297.

168 Marcotty, p.297.

169 Marcotty, p.297.

170 Marcotty, p.308.

171 Marcotty, p.297.

172 Marcotty, p.297.

173 Marcotty, p.297.

174 MacLennan, p.57.

175 Marcotty, p.297.

176 MacLennan, p.57-58.

177 Marcotty, p.312.

178 Cugini, p.10.

179 Marcotty, p.312.

180 Schneider, Michael G. and Steven W. Weingart and David M. Perlman, An Introduction to Programming and Problem Solving With Pascal, 2nd ed. (New York, New York: John Wiley & Sons, 1982), p.273.

181 Merchant, FORTRAN77 Language,p.229.

182 Merchant, FORTRAN77 Language,p.389.

183 Merchant, FORTRAN77 Language,p.389.

184 Merchant, FORTRAN77 Language,p.389.

185 Borland, p.137.

186 Brown, pp.82-83.

187 Borland, pp.147-148.

188 Kernighan, p.23.

189 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.120.

190 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.90.

191 Microsoft, QuickBASIC reference Manual, p.243.

192 Microsoft, QuickBASIC reference Manual, pp.480-482.

193 Microsoft, QuickBASIC reference Manual, p.242.

194 Microsoft, QuickBASIC reference Manual, pp.243-246.

195 Rojiani, Kamal, Associate Professor Civil Engineering, Personal Interview, 4/24/87.

196 Microsoft Corporation, telephone call, 4/24/87.

197 Schneider, Michael G. and Steven W. Weingart and David M. Perlman, An Introduction to Programming and Problem Solving With Pascal, 2nd ed. (New York, New York: John Wiley & Sons, 1982), p.299.

198 Marcotty, p.428.

199 Marcotty, p.428.

200 MacLennan, p.394.

201 MacLennan, p.394.

202 MacLennan, p.394.

203 Perrott, Ronald H. and Donald C.S. Allison Pascal for FORTRAN Programmers (Rockville, Maryland: Computer Science Press, Inc., 1984), p.187.

204 Pratt, Terrence W. Programming Languages Design and Implementation (Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1975), p.149

205 Pratt, p.149.

206 Pratt, p.152.

207 Pratt, p.153.

208 Pratt, pp.153-155.

209 Kernighan, p.3.

210 Jensen, Kathleen and Niklaus Wirth, Pascal User Manual and Report 3rd ed., (New York, New York: Springer-Verlag, 1985), p.113.

211 Marcotty, p.445.

212 Marcotty, p.445.

213 Marcotty, p.445.

214 Ghezzi, pp.196-197.

215 Marcotty, p.450.

216 Marcotty, p.450.

217 Marcotty, p.450.

218 Marcotty, p.450.

219 Marcotty, p.448.

220 Marcotty, p.448.

221 Marcotty, p.453.

222 Cugini, p.11.

223 Marcotty, p.455.

224 Marcotty, p.455.

225 Marcotty, p.455.

226 Marcotty, p.455.

227 Marcotty, p.455.

228 Marcotty, p.448.

229 Marcotty, p.454.

230 Microsoft, QuickBASIC reference Manual, pp.353

231 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.50.

DATA TYPE AND MANIPULATION

5.0 Introduction

> A program is useful because it models a real-world process. To do this, it manipulates abstract objects that represent real-world objects. The closer the properties of an abstract object mirror those of the corresponding real-world object, the more effective will be the program and the easier it will be to understand. [1]

Early languages allowed only numeric representations as abstract objects. An object is a term that denotes the storage area and value associated with a variable [2]. At that time, it was all that was needed since programs were largely used for numeric computations [3].

However, with the advances in programming languages and the new demands on programs, there came the need for new data types. The type of a identifier determines what type of value may be associated with the identifier [4]. Newer languages, such as Pascal, allow the user to define new data types. This allows the user extreme flexibility, but disciplines the user in terms of the preparation (variable declarations) that is required [5]. While the program becomes more readable due to the increased variable declarations, the cost of developing the code increases. Attempting to discuss all of the data types that could be created by the user would be an impossible task, even though some of these types may have distinct advantages (e.g., "subranges which limit the user to a particular domain" [6]). Thus, discussion on data types and their manipulation will be limited to the primitive types [7] (types that are directly supported by the languages). The following discussion includes type checking and coercion, elementary data (numeric, character, and logical), and aggregate data (arrays, files, records and sets) [8].

5.1 Checking and Coercion

"The partitioning of objects into types allows each assignment to be checked for a match between source value and target identifier. The validity of each operation for its operands can also be verified" [9]. Type checking is the ability of a processor to check for matching between the source value and its target identifier, as well as checking the validity of the operations using those values [10].

Constraint checking is similar to type checking. Constraint is defined as "the state of being checked, restricted, or compelled to avoid or perform some action" [11]. Thus constraint checking involves things like "...range constraints on integers[,]...index constraints on arrays[, etc.]" [12]. Unlike type checking, constraint checking is performed at execution time, since these properties cannot be checked until program execution [13]. User defined data types (which are not discussed in this thesis), such as subranges, are helpful in constraint checking [14].

Type checking can be done at various points in the code translation process. Type checking is static if it is done prior to execution. It is considered to be dynamic if it is performed during execution. "The essential difference between statically and dynamically typed languages is that in a statically typed language, the type is associated by declaration with an identifier. In a dynamically typed language, the type is associated with the value" [15]. Thus, dynamic typed languages require the type information to be stored along with the value [16]. Since dynamic type checking is done during execution, dynamically typed languages take longer to execute, and it is impossible to determine if a program contains type errors [17] before execution begins [18]. This reduces the reliability of the program [19].

Static type checking can lead to strongly typed languages. Strongly type languages allow type checking to be complete. This adds to the security of the language. Thus, "...a large number of errors can be detected before the program is run when it is feasible to make checks" [20]. Strongly typed languages allow a scalar variable to associate with scalar data only [21]. A variable may associate with data of only that type during its existence [22]. Operations are associated with particular data types [23]. The general mixing of data types as operands to these operations is not allowed [24]. Because this emphasizes discipline and control, it produces few surprising results [25] since the computer can perform extensive error checking [26].

In everyday regular non-computer calculations, numeric data is considered to be one type, a number [27]. The differentiation between integer, real, rationals, etc. is not made [28]. However, a computer considers each of these data types as separate entities [29]. At times, it is necessary to perform operations with different data types. "A Coercion is an implicit, context-dependent type conversion"[30]. Thus, type coercion allows a broader range of mixed mode expressions [31]. "Conversions from integer to fixed-point, from fixed-point to floating-point, and from floating-point to complex can generally be done without loss of information.

A coercion of this sort is called a widening" [32]. "There is no coercion from real to int[eger]" [33].

As the number of data types increase, so do the number of possible coercions [34]. However, some languages will not allow coercions without the use of specially designed functions (e.g., Pascal's use of the *ord* function to convert any scalar, except real, to an integer [35]) [36]. Thus, the programmer is forced to be aware of all conversions [37] which results in the possible reduction of inadvertent coercions, as well as rounding errors.

Strongly typed languages allow only a few, if any, coercions. "Errors can occur even in cases where the mixing of data types seems harmless" [38]. Thus, care must be taken to ensure that appropriate data types are used where they are expected.

"Pascal, because it allows the tag field of variant records to be changed, has exceptions to strong typing" [39]. Turbo Pascal is considered a strongly typed language [40]. Turbo Pascal [41], Microsoft FORTRAN [42], QuickBASIC [43], and C [44] have type checking, but allow certain coercions. However, they do not allow those that are plainly mistaken [45].

5.2 Elementary Data

"Elementary data types (also called scalar) are those which represent a single indivisible value,..." [46]. Elementary data types are also referred to as primitive types [47]. These data types include logical, character, numeric, and pointers.

5.2.1 Logical Type

The logical data type (also referred to as boolean types) is one of the simplest types in programming [48]. Logical data is data that is capable of containing only two values, TRUE or FALSE [49,50,51,52].

Various operations can be performed on this data type. The operators commonly are NOT, OR, and AND. NOT is defined as "a logic operator having the property that if P is a statement, then the NOT of P is true if P is false, false if P is true" [53]. OR is defined as "a logic operator having the property that if P is a statement, Q is a statement, R is a statement,..., then the OR of P,Q,R,..., is true if at least one statement is true, false if all statements are false" [54]. AND is defined as "a logic operator having the property that if P is a statement, Q is statement, R is a statement..., then the AND of P,Q,R..., is true if all statements are true, false if any statement is false" [55].

While QuickBASIC does not support user accessible logical data types [56], it does use the boolean operators [57] to "...perform tests on multiple relations,...or Boolean operations, and return a true (nonzero) or false (zero) value to be used in making a decision" [58]. Microsoft FORTRAN [59] and Turbo Pascal [60] support the boolean data type. C uses the numeric representations of zero and non-zero to represent true and false [61]. All the languages under consideration allow for combining conditions with the boolean operators (i.e., and, or, and not) [62,63,64,65].

5.2.2 Character Type

A character is defined as "A letter, digit, or other symbol that is used as part of the organization, control, or representation of data" [66]. Since the program communicates with the user through a sequence of characters, character manipulation is vital in programming languages [67]. "The real requirement is not to handle characters by themselves but to manipulate sequences of several characters. These sequences are usually known as *strings*" [68]. Two basic operations can be performed on strings. One operation considers the string in its entirety (e.g., comparison and printing) [69]. The other operation requires the string to be decomposed into substrings [70]. This differentiation can easily be seen in the two approaches used by languages to represent strings [71].

Some languages consider a string to be a primitive type [72]. As a primitive type, operations can be performed on strings (e.g. reference to substrings [73], printing strings, assignment to strings) [74].

Other languages do not consider the string as a primitive type [75], but represent "a string...as an array of characters" [76]. Thus, a string is an array that must be declared. As and array, only array operations are allowed on character strings, unless special functions are provided. Thus, substring references, printing of the string, and assignment to the string are not inherited in character arrays.

Of those languages that consider the string to be the primitive type, there are two methods commonly used in determining the allowable length of the string. Some will allow the string to be any length [77]. However, this method requires dynamic storage [78], unless it is predefined in the program [79]. Others require that the maximum length of the string be declared [80]. "When a string value is assigned to a string variable, the value will be padded on the right with blanks if it is too short or truncated if it is too long" [81].

All the languages under consideration allow a means to store characters [82,83,84,85]. Turbo Pascal [86] and C [87] represent a string of characters using an array. Furthermore, Turbo Pascal also has a special data type for strings (thus, the string is a primitive type) [88] that is compatible with the character type [89]. Microsoft FORTRAN [90], QuickBASIC [91], and Turbo Pascal [92] use a character string as a special type of aggregate (composed of smaller pieces [93]). The character string is of fixed length in Microsoft FORTRAN [94] and Turbo Pascal [95], while QuickBASIC [96] and C [97] allow variable length character strings. While standard "Pascal has very limited character manipulation[,] only assignment and comparison are provided" [98], Turbo Pascal offers various functions in character string manipulation [99].

Microsoft FORTRAN [100], QuickBASIC [101], and C [102] allow concatenation of characters into character strings. All the languages under consideration allow reference to a substring by a range expression [103,104,105,106].

QuickBASIC [107] allows for searching a string for the occurrence of another string. C allows for testing to see if a string is "...alphabetic, all upper or lower case, or numeric" [108].

Standard Pascal uses functions (e.g., ord and chr) for the conversion between characters and integers [109]. QuickBASIC provides conversion functions for this:

CVI converts a 2-byte string value to an integer value.
CVS converts a 4-byte string value to a single precision value.
CVD converts a 8-byte string value to a double precision value.
MKI$ converts an integer value to a string value.
MKS$ converts a single precision value to a string value).
MKD$ converts a double precision value to a string value [110].

While Microsoft FORTRAN [111] and C [112] do not have such functions, they can print converted data types via pseudo input-output statements (See Figure 5.1).

5.2.3 Numeric Types

Even Languages that are not mathematically based must use numeric data for counting, loops and control values. Thus, nearly all languages need some form of numeric data.

One of the differences between numeric data of different languages and pure mathematics is in their precision. A computer has finite storage elements. Thus, numeric data have finite representations. Mathematics can represent numbers with infinite precision.

Numeric data usually falls into three broad categories: integer, fixed-point, and floating-point. A brief discussion on the mathematical operands and intrinsic functions is also provided.

5.2.3.1 Integer Type

The integer type has a finite set of values [113]. An integer is defined as "[o]ne of the numbers zero, +1, -1, +2, -2..." [114]. The largest integer is defined by either the language definition or by the implementation.

Since integers can be represented exactly on most digital computers, the usual mathematical operations (i.e., addition, subtraction, and multiplication) can be performed without any consequences (except for underflow and overflow errors [115]). The relational operators (i.e., < and =) are usually defined over integers to yield a boolean result. Additional mathematical operators (i.e., mod and div) may also be included.

All of the languages under consideration support integers [116,117,118,119]. In addition, the normal mathematical and relational operators are defined over integers [120,121,122,123] (except raise to power in Pascal and C [124]).

In QuickBASIC, "Integers are stored internally as signed (two's compliment) 16-bit binary numbers ranging in value from -32,768 to 32,767" [125].

Microsoft FORTRAN has two integer types [126]. "A two-byte integer, INTEGER*2, can contain any value in the range -32,767 to 32767. A 4-byte integer, INTEGER*4, can contain any value in the range -2,147,483,647 to 2,147,483,647" [127].

"...In Turbo Pascal [integers] are limited to a range of -32768 to 32767. Integers occupy two bytes in memory" [128].

Microsoft C supports several types of integers by allowing the user to define types himself/herself (options include short, long, signed, unsigned, and any combination of these [129]). C's integer type, int, is not defined by the language [130]. The size corresponds to the natural size of an integer on a given machine (e.g., 16 bit or 32 bit) [131]. Remembering that

C is intended for systems programming, this allows a "...particular machine to handle integer values in the most efficient way for that machine. However, since the size of the int[eger]...varies, programs that depend on a specific int[eger] size may be nonportable" [132].

5.2.3.2 Fixed-point Type

A fixed-point number system is defined as "[a] radix numeration system in which the radix point is implicitly fixed in the series of digit places by some convention upon which agreement has been reached" [133]. Fixed-point arithmetic was used on early computer systems [134]. However, with the introduction of floating-point hardware, the fixed-point system fell into disuse [135]. It is still supported and used in applications that require an absolute bound on error [136]. Since fixed-point numbers are still represented in languages, they will be discussed.

"Values of fixed-point type are similar to integers in that they are uniformly spaced over a range. Two quantities are needed to specify this range:" [137] precision and scale. Precision is the total number of digits in the value [138]. Scale is the number of digits after the radix [139]. For example:

number	precision	scale
1.234	4	3
12.34	4	2
123.4	4	1
1234.	4	0
1.234567	7	6
12.34567	7	5
123.4567	7	4 [140].

Operations involving fixed-point numbers are more complex than those involving integers since precision and scale are fixed [141]. "The value must be copied into the target location so that its radix point is in the correct place" [142]. However, "The language must define what is to happen when the precision and scale of the assigned value do not match those of the target variable" [143]. "If the value has more fraction digits than the target, then the value must be truncated to fit, either with or without rounding. If the value has more digits before the point than the target, the situation is more serious" [144] since digits before the radix are involved (which makes rounding impractical) and the location of the radix is fixed. Further complications involve the operations of addition, subtraction, and multiplication. Any of these may produce a precision of the result that exceeds either operand [145]. Division is even more difficult to handle due to the possibility of requiring infinite precision to the right of the radix [146] in the quotient [147].

The key problem is centered around truncation and rounding [148]. The key question involves 'Where to truncate?' and 'When to round?'.

C, FORTRAN, and Pascal do not support fixed-point data type in any form (at least it couldn't be found in the manuals and available literature). QuickBASIC (not BASIC) supports the fixed-point data type only as a constant [149]. The constant "...can either be single-precision or double-precision [(See Section 5.2.3.3)]. Single-precision numeric constants are stored with 6 digits of precision (plus the exponent) and printed with up to 6 digits of precision. Double-precision numbers are stored with 14 digits of precision (plus the exponent) and printed with up to 14 digits of precision " [150].

5.2.3.3 Floating-point Type

"A floating-point value has two parts, a fraction [mantissa] and an exponent" [151]. The mantissa "...represents the significant figures of the value" [152] while "the exponent is a scaling factor to be applied to the fraction to obtain the proper value" [153]. For example:

number	mantissa	decimal representation
1.0E+01	1.0	10
1.234E+03	1.234	1234
1.234E-02	1.234	.01234
1.234E+00	1.234	1.234
1.234567E+04	1.234567	12345.67
1.234567E-03	1.234567	.001234567.

The programmer is primarily concerned with "...major characteristics of floating-point values...." [154]. Floating-point numbers contain a finite set of values [155]. Integers may not be included as a subset [156].

The floating-point number is referred to as the real type in some languages. In other languages it is considered the default type. However, it must be remembered that many floating-point representations cannot be represented accurately on the computer due to finite storage (e.g., pi, e and most decimal fractions [157]). "Repeated conversions between internal floating-point forms and external base-ten form may lead to considerable inaccuracies" [158].

Floating-point arithmetic has not been satisfactorily standardized [159]. Part of problem is that many floating-point data types do not include the integer as a subset due to deviations that arise in the floating-point arithmetic [160]. This contributes "...to the complexity and inaccuracy of floating-point computations" [161]. Many languages were unable to define the details of floating-point arithmetic because the designers of the languages did not want "...to specify machine capabilities that are not available on all machines" [162] (In addition, ANSI specifications do not allow it [163]). However, there are proposals to standardize the floating-point operations [164] (e.g., W. Stanley Brown [165] and IEEE [166]; IEEE has produced a standard, although it has errors [167]).

Some languages support extended precision by using something similar to double precision. Double precision appeared in the early 1960's [168]. Double precision is a floating-point number that contains twice as many digits as normal floating-point representations [169], and, therefore, provide increased accuracy.

All the languages under consideration support floating-point representations [170,171,172,173]. In addition C, FORTRAN, and QuickBASIC allow extended precision in floating-point representations [174,175,176]. Moreover, the normal mathematical and relational operators are defined over floating-point numbers [177,178,179,180]. Complex numbers are provided for only in Microsoft FORTRAN [181] (except as a user defined function [182]).

QuickBASIC supports two types of floating-point numbers. QuickBASIC's single precision "...numbers are accurate to 7 decimal places and have approximate ranges of -1.7E+38 to -2.9E-39 for negative values, true zero, and 2.9E-39 to 1.7E+38 for positive values" [183]. "Single precision numbers are stored internally as 32-bit (or 4-byte) binary numbers in sign-magnitude format" [184]. "Double-precision numbers have the same allowable ranges as single-precision numbers, but are accurate to 15 decimal places" [185]. The extended precision number uses 58 bits of memory.

Microsoft FORTRAN supports several forms of the floating-point number. The single precision IEEE real data type "...is normally an approximation of the real number desired and occupies four bytes [(32 bits)] of memory" [186]. The range of the IEEE single precision number is approximately 8.43E-37 to 3.37E+38 for positive real numbers, zero, and -3.37E+38 to -8.43E-37 for negative real numbers. "The precision is greater than six decimal digits and less than seven" [187].

Microsoft FORTRAN's double precision IEEE real data type "...is normally an approximation of the real number desired. A double precision real value can be a positive, negative, or zero value and occupies eight bytes [(32 bits)] of memory." [188]. The range of the IEEE extended precision number is approximately 4.19D-307 to 1.67D+308 for positive real numbers, zero, and -1.67D+308 to -4.19D-307 for negative real numbers [189]. "The precision is greater than 15 decimal digits" [190].

Microsoft FORTRAN's "decimal floating-point numbers consist of a byte containing a sign bit and a 7-bit exponent in excess 64 notation followed by a mantissa consisting of 6 (single) and 14 (double) binary coded decimal digits..." [191]. The ranges for the single precision decimal numbers are +1.0E-64 to +9.99E+62 for positive real numbers, zero, and -9.99E+62 to -1.0E-64 for negative real numbers [192]. The precision is exactly 6 digits [193]. The ranges for the extended precision decimal numbers are +1.0D-64 to +9.99D+62 for positive real numbers, zero, and -9.99D+62 to -1.0D-64 for negative real numbers [194]. The precision is exactly 14 digits [195].

Microsoft FORTRAN's "...COMPLEX*8 data type is an ordered pair of single precision real numbers with the second component representing an imaginary number. A COMPLEX*8 number occupies 8 bytes of memory" [196]. "A COMPLEX*16 data item consists of an ordered pair of double precision real numbers. COMPLEX*16 data items occupy 16 bytes of memory" [197]. "Each component (real and imaginary) of a COMPLEX*8 is a REAL*4. Each component of a COMPLEX*16 is a REAL*8" [198].

Pascal supports only one form of the floating-point number. The real number has a range of "...1E-38 to 1E+38 with a mantissa of up to 11 significant digits" [199].

Microsoft C supports two types of floating-point numbers. The float type uses the IEEE format [200]. Thus, the float type is composed of "...4 bytes, consisting of a sign bit, a 7-bit excess 127 binary exponent, and a 24-bit mantissa. The mantissa represents a number between 1.0 and 2.0" [201]. The float type may approximately range from 3.4E-38 to 3.4E+38 [202]. The double type (or long float type) is composed of "...8 bytes. The format is similar to the float format, except that the exponent is 11 bits excess 1023, and the mantissa has 52 bits,..." [203]. The double type may approximately range from 1.7E-308 to 1.7E+308 [204].

5.2.3.4 Mathematical Operations

"Expressions are built from operands (consisting of identifiers, integers, and parenthesized expressions) separated by operators. The operators are the symbols +, -, and *" [205]. These operators shall be called the base operators. Additional operators may include division [206], integer division [207,208], raise to power [209,210], and remainder after integer division (modulo arithmetic, mod) [211,212].

"If several operators occur in an expression, the parenthesized expressions are considered as a single operand" [213]. Unless otherwise grouped by parenthesis [214], operands are grouped together according to precedence [215]:

raise to power
multiplication, division
addition, subtraction, negation

Thus, operands used in addition and subtraction are treated equally and processed after multiplicative operations [216]. Groups of operands linked by operators of equal precedence are processed from left to right [217].

"Comparisons consist of parentheses enclosing a pair of operands, each an identifier or integer [(or character, real, etc.)], separated by one of the comparison operators =,..., <, and >" [218]. Some languages allow the use of floating-point numbers and/or characters [219], in addition to integers, to be enclosed by parentheses. Other languages may use different operators than those above (i.e., .GT. [220], != [221], \= [222], etc.).

All of the languages under consideration provide for the three basic arithmetic operations, division, and comparisons [223,224,225,226,227,228]. C offers special operations for incrementing and decrementing a variable [229] (e.g., ++i, --i, etc.). Turbo Pascal [230] and C [231] do not directly support raising to the power (depending upon the implementation, it is available through library functions). QuickBASIC [232] and Microsoft FORTRAN [233] provide many intrinsic functions (e.g., sine, cosine, square root, etc.). Pascal provides a smaller set of these functions [234]. C has no intrinsic functions [235,236] (depending upon the implementation, some are available through libraries).

5.2.4 Pointers

A pointer is a reference to another object [237]. In languages that support pointers, an object can be accessible through a chain of pointers (access path) [238]. "Two variables share an object if each has an access path to the object. A shared object modified via a certain access path [can make] the modification known to [other] access paths" [239]. While object sharing conserves storage, it may lead to programs that are difficult to understand since "...the values of variables can be modified even when they are not referenced" [240]. The objects connected to the pointer may change during the execution of the program.

In C, a variable may be of type pointer. They can be used in expressions, can appear on the left hand side of assignments, can be incremented and decremented to reference other objects, and can be manipulated as other variables can [241]. For the purpose of parameter passing only, a pointer can be considered as a formal parameter. However, any changes made to the pointer directly affect the actual parameter (See Figure 5.2).

Due to the importance of pointers in many programs, the rest of this section will provide a brief overview of pointers and their use. The rest of this section is not intended to be an integral part of this thesis, but is presented to offer more insight to the power of pointers.

Variables, up to this point, "...have been static, i.e. their form and size is pre-determined, and they exist throughout the entire execution of the block in which they are declared" [242]. Many programs need data structures that vary in form and size during execution. Dynamic variables serve this purpose since they may be generated and discarded within blocks of their declaration [243]. These dynamic variables must use pointers to be manipulated.

"Pointers can be used to create elegant, complex data structures" [244]. They can be used in C, as well as Pascal, to form linked lists. "This linked-list structure can be the basic foundation of stacks, queues, trees, and a host of other data structures, although a discussion of those structures is beyond the scope of this [thesis]" [245] (for a brief example in Pascal, see Figure 5.3).

"Since C is primarily a systems-programming language, directly accessing and manipulating memory addresses is one of C's basic concerns. Therefore, using pointer variables is an important part of C...." [246].

The responsibility of a programmer increases tremendously since he/she must remember if a identifier "...refers to an object directly or indirectly through a pointer" [247]. "The complex relationships that pointers are intended to represent are often very difficult to fathom" [248]. "Pointers have been lumped with the GOTO statement as a marvelous way to create impossible-to-understand programs" [249]. While "[t]his is certainly true when they are used carelessly,...with discipline...pointers can also be used to achieve clarity and simplicity" [250].

5.3 Aggregate Data

Aggregate is defined as "formed by the collection of units or particles into a body..." [251]. Thus, an aggregate data type is one that combines component values into a composite value [252].

Some "programming languages allow the programmer to specify aggregations of elementary data objects and, in turn, aggregations of aggregates" [253]. "Objects of a composite type [aggregate type [254]] are not indivisible, but have components bearing some relation to each other" [255]. Aggregate data, like other data, has a unique identifier associated with it [256]. A single elementary component of the aggregate can be manipulated at a time [257]. Some languages allow assignment and comparison of entire aggregates [258]. Aggregate data includes arrays, records, files, and sets [259].

5.3.1 Array Type

An array is one of the popular composite data types. An array is an elementary composite type "...where all the components are of the same type" [260]. "Arrays are ordered collections of data of similar type" [261].

"An array is basically a mapping from a range of contiguous integers to a set of elements" [262] (Pascal allows a range other than integers which will be briefly discussed later). There is a general form to the array structure:

array [t1] of t2 [263].

The t1 is called the domain [264] (index, subscript) and is usually type integer [265]. The t2 is referred to as the range (component) and is the type of each component of the array [266]. The difference between the upper and lower bounds of the subscripts defines the size of an array [267]. The domain must be specified by the programmer in most languages [268].

One of the simplest forms of "...an array is a representation of a table" [269]. Values of the table can be changed. "An important property of an array is that the value of any one of its elements can be changed without affecting the value of any of the other elements" [270].

There are several different classes of arrays based on when the range of the array is defined [271]. The first class determines the array size at the time of the array declaration [272]:

```
        A: array [1..5] of integer; [273].
```

Another class based on the same principle offers a little more variation [274]:

```
        N: integer constant =5;
        A: array [1..N] of integer; [275].
```

The last class concerns arrays in procedures. The size of the array is passed in via a parameter whose value was determined at the procedure invocation. This allows the size of the local array to vary in separate executions since the array size is determined at execution time [276]:

```
    procedure F (N:integer):
        A: array [1..N] of integer;  [277].
```

This type of an array is referred to as a dynamic array [278].

All of these classes were one dimensional (i.e., having only one subscript). They can also be referred to as vectors [279].

"Arrays can be nested so as to provide the effect of multidimensional entities" [280]. Most languages allow arrays to have multiple subscripts (dimensions) [281]:

```
B: array [1..5,1..10] of integer; [282].
A: array [1..5] of array [1..10] of integer;[283].
```

These two dimensional "...array[s] can be thought of as...rectangular arrangement[s] of elements with five rows and ten columns" [284].

> Conceptually, the elements of an array may be of any type, including simple types and other composite type, including arrays. In practice, most languages place restrictions on the types of elements, often restricting the elements to simple types [285].

A programmer can use an array without knowing how the elements are stored [286]. "The choice between...methods of storing is an implementation decision that does not affect the order of subscripts and only very rarely concerns the programmer" [287].

Some languages allow reference to "...a subpart of an array as a single entity" [288]. Thus a reference such as

```
B[2..8]
```

would reference element 2 to 8 of array B [289]. This subpart is also referred to as a trimmed reference [290] or slice [291]. This process is essential for languages that use arrays to represent strings to allow proper access to substrings [292].

Every language that supports arrays allow assignment to arrays [293]. Some of the languages allow complete arrays to be assigned in one statement. Thus, a statement such as

```
A:=B;
```

where both A and B are 10-element arrays, would assign the contents of B to A [294]. Some languages also use a process similar to above to initialize arrays [295]. "Most languages, however, allow assignment only on an element-by-element basis..." [296]:

```
A[0] :=10;
A[1] :=20;
A[2] :=30;
A[3] :=40;
A[4] :=50;
A[5] :=60;
A[6] :=70;
A[7] :=80;
A[8] :=90;
A[9] :=100;
A[10]:=1000;
```

All of the languages under consideration support arrays [297,298,299,300]. C [301,302] and Microsoft FORTRAN [303] (not FORTRAN77 [304]) fix the lower bound of the indexes (0 and 1, respectively). Quickbasic will allow the choice between zero and one for the lower bound [305]. Turbo Pascal allows the user to specify the full range of the subscripts [306].

Only Pascal allows the subscripts to be some entity other than an integer [307,308]. Thus, the index may be any of the scalar types [309], and the following declarations are possible:

```
type
  Players = (Player1,Player2,Player3,
             Player4);
  Hand    = (One,Two,Pair,TwoPair,Three,
             Straight,Flush,FullHouse,Four,
             StraightFlush,RSF);
  LegalBid = 1..200;
  Bid     = array[Players] of LegalBid;
var
  Player  : array[Players] of Hand;
  Pot     : Bid;
```

Assuming the above declarations, the following assignments could be made:

```
Player[Player3] := FullHouse;
Pot[Player3]    := 100;
Player[Player4] := Flush;
Pot[Player4]    := 50;
```

Convenience should allow for initialization of arrays by some method other than individual assignment to each element. C [310] and Turbo Pascal [311] allow this with a list of values in declaration statement. Microsoft FORTRAN [312] and QuickBASIC [313] use the DATA statement to assign values to an array via nested loops. Turbo Pascal allow the user to copy arrays with a single assignment statement [314].

Turbo Pascal [315] and C [316] store arrays in row major order while Microsoft FORTRAN [317] stores arrays in column major order. QuickBASIC will allow the user to choose row or column order [318].

5.3.2 Records

Records are structures that can consist of components (fields [319]) of potentially different types [320,321]. "Each component has a name and a value" [322]. In addition, each component may be another aggregate data type (e.g., an array or another record) [323].

By studying figure 5.4, one can see the general declarations for a driver's license. "The basic operation on record types is component selection" [324]. Thus, the identifier LICENSE.DRIVING_CODE refers to the driving code component of the drivers license, LICENSE.DRIVER.LAST_NAME refers to the last name of the driver, and LICENSE.LICENSE_NUM refers to the string containing the license number [325]. "A reference to the component of a record behaves just as a reference to the component of an array" [326]. Thus, assignment is done in the same fashion as an array:

```
LICENSE.DRIVER.FIRST_NAME    :='DAVID'
LICENSE.DRIVER.MIDDLE_INITIAL:='C'
LICENSE.DRIVER.LAST_NAME     :='HUXFORD' [327].
```

Pascal offers an alternative form of assignment to save typing on the programmers part:

```
with LICENSE.DRIVER do
  begin
   FIRST_NAME    := 'ELIZABETH';
   MIDDLE_INITIAL := 'L';
   LAST_NAME     := 'HUXFORD'
  end; [328,329].
```

Arrays and records differ on two major points. An array is homogeneous (all components are the same type), while a record is heterogeneous (all components are not necessarily the same type) [330]. The other difference concerns the manner in which the components can be selected. An array's component can be computed. The index can be computed by some mathematical expression. "This is an important features since it allows, for example, writing a loop that processes all the elements of an array" [331]. This cannot be done with records. Thus, an array's selector (i.e., subscript) is dynamic, while a records selector (i.e., LICENSE.DRIVER.FIRST_NAME) is static.

At times, there are cases when the information is not available or when additional information is needed [332]. Considering the drivers license, "...the driver may not have a middle initial, and the driving code may indicate a special or restricted permit requiring other information" [333]. These special cases can be handled using variant records.

A variant record must have a tag (discriminant [334]) and some sort of selection for the different values of the tag. To build more on the drivers license using variant records, see Figure 5.5. An additional variant record, this time dealing with airplanes, is supplied in Figure 5.6.

The same rules for selection apply to variant records. However, care must be taken. If the tag DRIVING_CODE is 'S', "[I]t would be an error to reference LICENSE.DAYLIGHT_ONLY. This field does not exist when the value of the tag is 'S'. It only exists when the tag's value is 'R'" [335].

"Variant records, while useful, raise some difficulties with reliability and implementation" [336]. Assignments to a tag field can be troublesome. The tag value is determined when the record object is created. Assigning a new value to the tag "...implies a change in the structure of the record" [337]. Thus, a change in storage could be needed. "...[D]ynamic checks are...required whenever a reference is made, and it seems likely that most systems will omit these checks in the name of efficiency" [338].

An additional problem concerns readable syntax. Referring back to the last driver's license example will show "...that DRIVER has potentially four components" [339]. However, just by the identifier DRIVER, it is not obvious that these components exist.

Only Microsoft FORTRAN [340] and QuickBASIC [341] do not support the record type directly. C has very limited record operations [342]. Pascal fully supports records and variant records [343]. In addition, records of similar type may be copied in a single assignment statement similar to the same process with arrays [344,345].

5.3.3 Files

"Files are organized collections of data which may exist outside the program" [346]. There are two basic types of files; Sequential access files (SASD [347]) and direct access files (DASD [348],Random access files) [349].

Sequential access is defined as "[t]he facility to obtain data from a storage device or to enter data into a storage device in such a way that the process depends on the location of that data and on a reference to data previously accessed" [350]. Sequential files are "...viewed as a sequence of values that are transferred in the order of their appearance (as produced by the program or by the environment). When the file is opened, transfer starts from the beginning of the file" [351]. This data can be accessed only with respect to the current position in the file [352]. Thus, data must be input from beginning to end, and input files can only be read in the order in which they were written [353]. "The basic operation is the read next operation, which gives the next data item in order" [354]. It is not possible to access sequential files from the middle of the data. "Generally, sequential files can only be opened for read mode or for write mode and cannot be used for a mixture of these modes without intermediate closing and reopening. Text files are almost always sequential access files" [355]. All the languages under consideration support sequential files [356,357,358,359].

Direct access is defined as "[t]he facility to obtain data from storage devices or to enter data into a storage device in such a way that the process depends only on the location of that data and not on a reference to data previously accessed" [360]. Direct access files are "...viewed as a set of elements occupying consecutive positions in linear order; a value can be transferred to or from an element of the file at any selected position" [361]. Each record (i.e., a set of data or words treated as a unit [362]) [363] is numbered sequentially, beginning with 1 [364]. Each record may be accessed by means of the identifying number, also called a key [365]. "Each statement that accesses a record in random a access file must provide the key to identify the record" [366]. Microsoft FORTRAN [367,368] and QuickBASIC support direct files [369]. While standard Pascal does not allow direct access, Turbo Pascal does [370].

"Between these two forms of access, sequential and random, there are many intermediate forms of file organization" [371]. However, a discussion of file organization goes beyond the scope of this thesis.

Files can be considered either internal or external. Internal files usually consist of aggregate data with related components organized in specified formats [372]. External files contain data that is input or output from anywhere outside (external) of the program [373]. Elements of external files "...are usually strings of characters, called lines, in either fixed or free format" [374]. Fixed format fixes the representation of data within the line. Free format allows data formatting to be done automatically [375].

C is the only language that does not allow internal files [376]. C [377] and Microsoft FORTRAN [378] allow both fixed and free on I/O external files. Pascal [379] and QuickBASIC [380] do not have fixed format on input from external files.

QuickBASIC [381], Turbo Pascal [382], and Microsoft FORTRAN [383] support both sequential and random access files. Microsoft C reads characters and outputs character one character at a time, and, therefore, uses only sequential files.

Only FORTRAN does not "...provide a way to read an entire line (whose length is not known in advance) from a terminal as a single character string" [384]. C and Pascal can read or write part of a line as a single operation. BASIC can only write partial lines [385]. FORTRAN can not handle partial lines at all [386].

5.3.4 Sets

A set is defined as "[a] finite or infinite number of objects of any kind, of entities, or of concepts, that have a given property or properties in common" [387]. For the purposes of this thesis "[a] set is an unordered collection of elements chosen form a specified universe" [388]. "A set is a data aggregate that contains an unordered collection of distinct values" [389].

"The usual operations associated with sets are testing whether an element belongs to a set, inserting or deleting an element, testing whether one set is a subset of another, and taking the union, intersection, difference, or negation of sets" [390].

A set can be declared by the following statement:

```
var S,T: set of 1..10 [391].
```

Thus the identifiers S and T now can contain any one of, or group of, the values 1 to 10 [392]. A set can be initialized by the following statements [please, try not confuse footnotes marks '[]' with set notation]:

```
S := [1,2,3,5,7]
T := [1,2,3,4,5,6] [393].
```

Furthermore, intersections (* [394]) and unions (+ [395]) can also be performed:

```
T := S * T [396].
```

Thus, T is now associated with the value [1,2,3,5]. This can also be used in comparisons.

Typical comparison operators for sets may include:

1) = Equality
2) < > Inequality
3) > = True if all members in the second operand are included in the first operand
4) < = True if all members in the first operand are included in the second operand
5) IN Test on set membership. The second operand is of set type, and the first operand is an expression of the same type as the base type of the set. The result is true if the first operand is a member of the second operand, otherwise it is false [397].

For a brief example of sets see Figure 5.7.

Only Turbo Pascal (and standard Pascal) supports the set type [398]. QuickBASIC [399] and Microsoft FORTRAN [400] do not support this type at all. C uses bitwise operations on integers along with its ability to define bit-fields to roughly simulate sets. Turbo Pascal [401] supports set difference and subset testing (not negation) while C [402] supports set negation (not difference or subset testing).

5.4 Conclusion

The type of a variable determines what type (i.e., integer, floating point, etc.) of value may be associated with the variable. It is important that languages be able to support and check various data types.

Strict static type checking reduces the possibility of errors in identifier names. Strict type checking also requires increased identifier declarations which increases the readability and understandability of the program. This will increase the security of a program. Pascal and C have strict type checking, while QuickBASIC and FORTRAN allow many implicit entities and have little type checking.

Coercions usually occur without the knowledge of the programmer, which is not considered a good programming practice. Coercions allows a broader range of mixed mode operations. As the number of data types increase, so do the number of possible coercions. Coercions are allowed in all of the languages under consideration (QuickBASIC, FORTRAN, Pascal, and C in order of the number of coercions allowed).

Logical data is useful for the efficient testing of conditions. Therefore, logical data should be supported by the languages. Logical data is directly supported by Microsoft FORTRAN and Turbo Pascal. It is indirectly supported by Microsoft C. Microsoft QuickBASIC does not support logical data.

Character data is necessary to communicate with the user. All of the languages under consideration support character data. However, QuickBASIC and FORTRAN support the string as the base character type, while Pascal and C consider an individual character as the base character type. Thus, substring operations and string manipulations are easier to implement and more accessible in QuickBASIC and FORTRAN than in Turbo Pascal and C. This can become crucial to programmers who use string manipulations.

A language must support numeric data for counting, loops and control variables. Mathematically based languages need numeric data to perform calculations. A language that does not support these types is at a severe disadvantage in terms of mathematical capabilities when compared to those languages that do support these types. All the languages under consideration support numeric data. The integer type and the floating-point type are supported in some form by each of the considered languages. The fixed-point data type is supported by QuickBASIC only. Even though floating-point arithmetic has not been standardized for computer applications, it is still used in many languages today. An extension of the floating-point type are complex numbers, supported only by FORTRAN, and double precision, supported by all of the considered languages except Turbo Pascal. The floating-point data type, with increased precision, and the integer data type are extensively used in engineering applications. All of the languages under consideration support the three basic mathematical operations of addition, subtraction, and multiplication as well as division.

Intrinsic functions can make a programmer's task easier by not requiring the programmer to code these functions himself/herself. Thus, languages that supply intrinsic functions are advantageous to those that do not. In terms of engineering applications, the more mathematical operations that are available, the more uses the language may have. Intrinsic functions are supplied by QuickBASIC, Microsoft FORTRAN, and Turbo Pascal (C has no intrinsic functions as a standard). More of these functions can be obtained through libraries available in a variety of implementations.

Pointers are variables that contain addresses of other data objects. One of the basic operations of pointers is used by C to emulate reference parameters. A form of pointers may be used in FORTRAN to create a form of dynamic arrays. Pointers are needed in Turbo pascal to extend its data area past 64K. Pointers are supported by Pascal and C only (although FORTRAN uses pointers implicitly in subroutines). In addition to this basic use, pointers are used extensively in lists, linked list, and trees. If the programmer needs any structure such as these, the language's support of pointers will be helpful.

Arrays can be used by the programmer without knowing how the elements are stored. Arrays are used in engineering applications for a variety of purposes. Multidimensional arrays are used frequently to represent matrices and other mathematical structures. Thus, their support in programming languages is mandatory. All of the languages under consideration support arrays. Assignments to the array, the ability to change the elements of an array, and the use of arrays in mathematical expressions are just as important as the existence of the array in the language. In addition, arrays can be useful in the storing of character data. Some languages, such as C and Pascal, represent strings with the array. Each element of the array holds a corresponding character of the string. Thus string manipulation is done through arrays. In order for this to be accomplished, languages should also support access to substrings.

The record data type is a complex structure that is used in many environments. However, it is commonly not used in engineering applications. Thus, a language's support of the record data type is not crucial in engineering applications. Only Turbo Pascal fully supports the record data type. Microsoft C has limited record manipulations, while Microsoft FORTRAN and QuickBASIC do not support the record type at all.

Files are used in many applications. Files enable a programmer to store large amounts of data without having to retype that data each time a program is executed. Thus, a program can access data directly from an input file, and store the results in an output file. Thus, a variety of input files, with slightly altered data, can be used by a program to produce a variety of output files which can later be studied to determine the effects of the altered input. An efficient use of files will free the user and allow him/her to do other tasks while the program automatically inputs the data and output the results.

The above statements assume the use of external files. External files are preferable, since the user does not need to know the actual code of a program to modify data. All the user needs to know is the format of data input. If internal files are used, the user would need to access the source code, modify the data, and recompile the program. Programs that require the use of altered data to perform tasks should not use internal files. All of the languages support external files. Only C will not allow internal files.

The set type is not commonly used in engineering applications. However, this is probably due to the fact that the languages commonly used in engineering applications do not support the set type. One of the most important operations that a set could be used for is membership (i.e., subrange checking, checking for allowable input, etc.). Other operations involve union, negation, and difference of sets. Sets are supported directly by Turbo Pascal only.

C Program Listing:

```
main()
{
  int  x;                /* x is an integer   */
  char y;                /* y is a character */
  x=65
  y='A'
  printf ("%c\n",x);    /* a character is printed */
  printf ("%d\n",y);    /* an integer is printed   */
```

Output:

```
   A
   65
```

Example adapted from: Traister, Robert J. Going From BASIC to C (Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1985), p.28.

FORTRAN Program Listing:

```
      integer  x,y,z
      x=65
      y=67
      z=69
C
C x, y, and z are integers printed out as characters
C
      write (*,'(1x,a,2x,a,2x,a)') x,y,z
      stop
      end
```

output:

```
 A  C  E
```

FIGURE 5.1 - PSEUDO OUTPUT

INCORRECT IMPLEMENTATION IN C

```
swap(a,b);

Where swap is defined as:

swap(x,y); /* WRONG */
int x,y;                   x and y are local.
{
  int temp;                Here, swap cannot find the
  temp=x;                    arguments a and b in the
  x=y;                       routine that called it.
  y=temp;
}
```

CORRECT IMPLEMENTATION IN C

```
swap(&a,&b);

Where swap is defined as:

swap(x,y); /* interchange *px and *py */
int *px,*py;                   x and y are local.
{
  int temp;                Here, & gives the address of
  temp=*px;                    the variables.  *px and
  *px=*py;                     *py are the pointers
  *py=temp;                    that contain the variables
}                              addresses.
```

Figure taken from: Kernighan, Brian W. and Dennis M. Ritchie The C Programming Language (Englewood Cliffs, New Jersey: Prentice-Hall, Inc. 1978.) pp.91-92.

FIGURE 5.2 - POINTERS IN C

```
procedure Jobs;
(*
  This program demonstrates the use of pointers to
    maintain a list of names and related job
    descriptions.  Names and job desires will be read
    until a blank name is entered.  Then the entire
    list is printed.  Finally, the memory used by the
    list is released for other use.  The pointer
    variable HeapTop is used only for the purpose of
    recording and storing the initial value of the heap
    pointer.

DECLARATIONS
*)


type
  PersonPointer=^PersonRecord;            (* pointer *)
  PersonRecord = record                   (* record  *)
                  Name : string[50]
                  Job  : string[50]
                  Next : PersonPointer;
              end;
var
  HeapTop : ^Integer;                     (* pointer *)
  FirstPerson, LastPerson, NewPerson:PersonPointer;
  Name: string[50];
```

FIGURE 5.3 - POINTERS IN A LIST IN PASCAL

```
begin
     FirstPerson:=nil;
  Mark(Heaptop);
  repeat                      (* until blank name entered *)
    Write('Enter name:        ');
    Readln(Name);                         (* read name *)
    if Name<>'' then
    begin
      New(NewPerson);
      NewPerson^.Name :=Name;
      Write('Enter profession: ');
      Readln(NewPerson^.Job);             (* read job *)
      Writeln;
      if FirstPerson =nil then
        FirstPerson := NewPerson
      else
        LastPerson^.Next :=NewPerson;
      LastPerson:=NewPerson;
      LastPerson^.Next:=nil;
    end;
  until Name="";
  while FirstPerson<>nil do              (* print list *)
  with FirstPerson^ do
  begin
    Writeln(Name,' is a ',Job);
    FirstPerson:=Next;
  end;
  Release(Heaptop);                     (* release memory *)
end.
```

Example adapted from: Borland International, Turbo Pascal Reference Manual (Scotts Valley, California: Borland International, Inc., 1986), p.122-123.

FIGURE 5.3 (CONT) - POINTERS IN A LIST IN PASCAL

General form desired:

```
      Driver :a name consisting of a
           First name    : a string of letters
           Middle initial: a single letter
           Last name     : a string of letters

           License number:  eight digits

           Expiration date: calendar date composed of a
                   Month : a number from 1 to 12
                   Day   : a number from 1 to 31
                   Year  : a four digit number

      Driving Code: a character
```

```
Declarations to obtain above form:

LICENSE: record
        DRIVER: record
           FIRST_NAME    : string;
           MIDDLE_INITIAL: string;
           LAST_NAME     : string;
        end record;
        LICENSE_NUM: string;
        EXPIRATION_DATE: record
           MONTH: integer;
           DAY  : integer;
           YEAR : integer;
        end record;
        DRIVING_CODE: string;
end record;
```

Figure taken from: Marcotty, Michael and Henry F. Ledgard Programming Language Landscape 2nd ed. (Chicago, Illinois: Science Research Associates, Inc., 1986) pp.248-249.

FIGURE 5.4 - RECORD DECLARATION

```
LICENSE : record
   DRIVER: record
      FIRST_NAME: string;
      LAST_NAME : string;

      HAS_MIDDLE_INITIAL: boolean;   --tag
      case HAS_MIDDLE_INITIAL of
        when true  => MIDDLE_INITIAL: string;
        when false => null;
      end case;
   end record;

   LICENSE_NUM: string;

   DRIVING_CODE: string      --tag
   case DRIVING_CODE of

      when 'S' => record     --special vehicle
         VEHICLE_TYPE    :integer;
         PASSENGER_PERMIT:boolean;
         ZONE_CODE       :boolean;
      end record;

      when 'R' => record     --restricted permit
         CORRECTIVE_LENSES:boolean;
         DAYLIGHT_ONLY    :boolean;
         AUTO_TRANSMISSION:boolean;
      end record;

      else => NULL;
   end case;
   EXPIRATION_DATE: record
      MONTH: integer;
      DAY  : integer;
      YEAR : integer;
   end record;
end record;
```

Figure taken from: Marcotty, Michael and Henry F. Ledgard Programming Language Landscape 2nd ed. (Chicago, Illinois: Science Research Associates, Inc., 1986) p.253.

FIGURE 5.5 - VARIANT RECORD DECLARATION

```
type plane = record
  flight : 0..999;
  kind   : (B727,B737,B747)

  case status: (inAir, onGround, atTerminal) of

  inAir: (
    altitude    : 0..100000;
    heading     : 0..359;
    arrival     : time;
    destination : airport );

  onGround: (
    location : airport;
    runway   : runwayNumber );

  atTerminal: (
    parked    : airport;
    gate:     : 1..100;
    departure: time )

end {plane};
```

Figure taken from: MacLennan, Bruce J Principles of Programming Languages: Design, Evaluation, and Implementation (New York, New York: CBS College Publishing, 1983) p.205.

FIGURE 5.6 - VARIANT RECORD DECLARATION

```
(* This program reads a character from some source.
   It determines if that character in is one of the
   sets digit, letter, or punctuation.  If the
   character is in one of those sets, program control
   is transferred to the corresponding section.  If
   the character is not in any of those sets, then the
   character is illegal, and must be handled.  *)


var
   digits, letters, punctuation: set of char;
   ch: char;

begin
  digits  := ['0','1','2','3','4','5','6','7','8','9'];
  letters := ['a','b','c',...,'z','A','B',...,'Z'];
  punctuation := ['(',')','.',',',';',':','!',];
   ...
  read(ch);

  if ch in letters then ...handle letters
  else if ch in digits then ... handle digits
  else if ch in punctuation then ... handle punctuation
  else ... handle illegal characters
   ...

end
```

Figure adapted from: MacLennan, Bruce J Principles of Programming Languages: Design, Evaluation, and Implementation (New York, New York: CBS College Publishing, 1983) p.194.

FIGURE 5.7 - SETS

NOTES - CHAPTER 5

1 Marcotty, p.229.

2 Ghezzi, p.54.

3 Marcotty, p.229.

4 Ghezzi, p.55.

5 Cugini, p.15.

6 Lee, Personal Interview, 4/30/87.

7 Marcotty, p.229.

8 Cugini, p.vi-vii.

9 Marcotty, p.254.

10 Marcotty, p.254.

11 Webster's New Collegiate Dictionary, p.241.

12 Ghezzi, p.129.

13 Lee, Personal Interview, 5/1/87.

14 Lee, Personal Interview, 5/1/87.

15 Marcotty, p.255.

16 Marcotty, p.255.

17 Marcotty, p.255.

18 Lee, Personal Interview, 4/30/87.

19 Marcotty, p.255.

20 Marcotty, p.255.

21 Schneider, Michael G. and Steven W. Weingart and David M. Perlman, An Introduction to Programming and Problem Solving With Pascal, 2nd ed. (New York, New York: John Wiley & Sons, 1982), p.87.

22 Schneider, p.87.

23 Schneider, p.87.

24 Schneider, p.87.

25 Cugini, p.15.

26 Schneider, p.87.

27 Marcotty, p.244.

28 Marcotty, p.244.

29 Marcotty, p.244.

30 MacLennan, Bruce J. Principles of Programming Languages: Design, Evaluation, and Implementation (New York: CBS College Publishing, 1983), p.68.

31 Maclennan, p.68.

32 Marcotty, p.244.

33 Ghezzi, p.133.

34 Marcotty, p.244.

35 Borland, p.142.

36 Marcotty, p.244.

37 Marcotty, p.244.

38 Schneider, p.96.

39 Marcotty, p.255.

40 Schneider, p.87.

41 Borland, p.51.

42 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.33.

43 Microsoft, QuickBASIC reference Manual, p.178.

44 Kernighan, p.3.

45 Cugini, p.15.

46 Cugini, p.16.

47 Marcotty, p.237.

48 Marcotty, p.237.

49 Microsoft, Microsoft FORTRAN Compiler Reference Manual, pp.27.

50 Cugini, p.18.

51 Marcotty, p.237.

52 Ghezzi, p.55.

53 American National Standards Committee X3, p.90.

54 American National Standards Committee X3, p.95.

55 American National Standards Committee X3, pp.5-6.

56 Microsoft, QuickBASIC reference Manual, pp.161-162.

57 Microsoft, QuickBASIC reference Manual, p.173.

58 Microsoft, QuickBASIC reference Manual, p.173.

59 Microsoft, Microsoft FORTRAN Compiler Reference Manual, pp.27.

60 Borland, pp.42.

61 Kernighan, p.41.

62 Kernighan, p.189.

63 Borland, pp.51-54.

64 Microsoft, Microsoft FORTRAN Compiler Reference Manual, pp.36-37.

65 Microsoft, QuickBASIC reference Manual, pp.173-176.

66 American National Standards Committee X3, p.19.

67 Marcotty, p.238.

68 Marcotty, p.238.

69 Marcotty, p.238.

70 Marcotty, p.238.

71 Marcotty, p.238.

72 Marcotty, p.238.

73 Marcotty, p.238.

74 Lee, Personal Interview, 4/30/87.

75 Marcotty, p.239.

76 Marcotty, p.239.

77 Marcotty, p.239.

78 Marcotty, p.239.

79 Lee, Personal Interview, 4/30/87.

80 Marcotty, p.239.

81 Marcotty, p.239.

82 Kernighan, p.193.

83 Borland, p.44.

84 Microsoft, QuickBASIC reference Manual, p.161.

85 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.34.

86 Borland, p.44,67.

87 Kernighan, p.181.

88 Borland, pp.67-73.

89 Borland, p.73.

90 Microsoft, Microsoft FORTRAN Compiler Reference Manual, pp.37-28.

91 Microsoft, QuickBASIC reference Manual, pp.177-178.

92 Borland, p.67.

93 Cugini, p.21.

94 Merchant, FORTRAN77, p.147.

95 Borland, p.59.

96 Microsoft, QuickBASIC reference Manual, p.177.

97 Kernighan, pp.84,109,124.

98 Cugini, p.18.

99 Borland, pp.67-72.

100 Merchant, FORTRAN77, p.151.

101 Microsoft, QuickBASIC reference Manual, pp.177-178.

102 Kernighan, p.44.

103 Borland, p.72.

104 Merchant, FORTRAN77, pp.151-152.

105 Microsoft, QuickBASIC reference Manual, pp.,322,345-346,435.

106 Microsoft, Microsoft C Compiler, The Run-Time Library Reference Manual (Redmond, Washington: Microsoft Corporation, 1986), p.69.

107 Microsoft, QuickBASIC reference Manual, pp.,322,345-346,435.

108 Cugini, p.18.

109 Schneider, p.102.

110 Microsoft, QuickBASIC reference Manual, p.236,348.

111 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.155.

112 Traister, p.28.

113 Marcotty, p.241.

114 American National Standards Committee X3, p.67.

115 Lee, Personal Interview, 4/30/87.

116 Kernighan, p.193.

117 Borland, p.41.

118 Microsoft, QuickBASIC reference Manual, p.161.

119 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.33.

120 Kernighan, pp.38-39.

121 Borland, p.53.

122 Microsoft, QuickBASIC reference Manual, p.173.

123 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.35.

124 Lee, Personal Interview, 4/30/87.

125 Microsoft, QuickBASIC reference Manual, p.161.

126 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.22.

127 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.22.

128 Borland, p.41.

129 Microsoft, Microsoft C Compiler, Microsoft Code View and C Language Reference Manual (Redmond, Washington: Microsoft Incorporated, 1986), p.42-46.

130 Microsoft, Microsoft C Compiler, Microsoft Code View and C Language Reference Manual, p.45.

131 Microsoft, Microsoft C Compiler, Microsoft Code View and C Language Reference Manual, p.45.

132 Microsoft, Microsoft C Compiler, Microsoft Code View and C Language Reference Manual, p.45.

133 American National Standards Committee X3, p.67.

134 MacLennan, p.274.

135 MacLennan, p.274.

136 MacLennan, p.274.

137 Marcotty, p.241.

138 Marcotty, p.241.

139 Marcotty, p.241.

140 Marcotty, p.241.

141 Marcotty, p.241.

142 Marcotty, p.241.

143 Marcotty, p.242.

144 Marcotty, p.242.

145 Marcotty, p.242.

146 Lee, Personal Interview, 4/30/87.

147 Marcotty, p.242.

148 Marcotty, p.242.

149 Microsoft, QuickBASIC reference Manual, p.163.

150 Microsoft, QuickBASIC reference Manual, p.164.

151 Marcotty, p.242.

152 Marcotty, p.243.

153 Marcotty, p.243.

154 Marcotty, p.243.

155 Marcotty, p.243.

156 Marcotty, p.243.

157 Lee, Personal Interview, 4/30/87.

158 Marcotty, p.243.

159 Marcotty, p.243.

160 Marcotty, p.243.

161 Marcotty, p.243.

162 Marcotty, p.243.

163 Lee, Personal Interview, 4/30/87.

164 Marcotty, p.243.

165 Brown, W. Stanley, A simple but realistic model of floating-point computation, in ACM Transaction on Mathematical Software, vol.7, no.4, December 1981, pp.445-480.

166 IEEE, A proposed standard for binary floating-point arithmetic, Draft 8.0 of IEEE Task P754 with introductory comments by David Stevenson, in Computer, Vol.14, No.3, March 1981, pp.51-62.

167 Lee, Personal Interview, 4/30/87.

168 MacLennan, p.65.

169 MacLennan, p.65.

170 Kernighan, p.193.

171 Borland, p.42.

172 Microsoft, QuickBASIC reference Manual, p.161.

173 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.33.

174 Kernighan, p.193.

175 Microsoft, QuickBASIC reference Manual, p.162.

176 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.33.

177 Kernighan, pp.38-39.

178 Borland, pp.53.

179 Microsoft, QuickBASIC reference Manual, p.173.

180 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.35.

181 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.33.

182 Lee, Personal Interview, 4/15/87.

183 Microsoft, QuickBASIC reference Manual, p.161.

184 Microsoft, QuickBASIC reference Manual, p.161.

185 Microsoft, QuickBASIC reference Manual, p.162.

186 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.23.

187 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.23.

188 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.25.

189 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.25.

190 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.25.

191 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.26.

192 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.26.

193 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.26.

194 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.26.

195 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.26.

196 Microsoft, <u>Microsoft FORTRAN Compiler Reference Manual</u>, p.26.

197 Microsoft, <u>Microsoft FORTRAN Compiler Reference Manual</u>, p.27.

198 Microsoft, <u>Microsoft FORTRAN Compiler Reference Manual</u>, p.27.

199 Borland, p.42.

200 Microsoft, <u>Microsoft C Compiler, Microsoft Code View and C Language Reference Manual</u>, p.46.

201 Microsoft, <u>Microsoft C Compiler, Microsoft Code View and C Language Reference Manual</u>, p.46.

202 Microsoft, <u>Microsoft C Compiler, Microsoft Code View and C Language Reference Manual</u>, p.46.

203 Microsoft, <u>Microsoft C Compiler, Microsoft Code View and C Language Reference Manual</u>, p.46.

204 Microsoft, <u>Microsoft C Compiler, Microsoft Code View and C Language Reference Manual</u>, p.46.

205 Marcotty, p.28.

206 Merchant, <u>FORTRAN77</u>, p.69.

207 Microsoft, <u>QuickBASIC reference Manual</u>, p.171.

208 Marcotty, p.241.

209 Microsoft, <u>QuickBASIC reference Manual</u>, pp.170-171.

210 Merchant, <u>FORTRAN77</u>, p.69.

211 Microsoft, <u>QuickBASIC reference Manual</u>, p.172.

212 Marcotty, p.241.

213 Marcotty, p.28.

214 Merchant, <u>FORTRAN77</u>, p.71.

215 Merchant, <u>FORTRAN77</u>, p.71.

216 Marcotty, p.28.

217 Marcotty, p.28.

218 Marcotty, p.29.

219 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.35.

220 Merchant, FORTRAN77,p.100.

221 Kernighan, p.190.

222 Lee, Personal Interview, 4/30/87.

223 Kernighan, pp.187-192.

224 Schneider, pp.95-96.

225 Microsoft, QuickBASIC reference Manual, pp.169-176.

226 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.32.

227 Borland, p.51.

228 Merchant, pp.69,100.

229 Kernighan, p.187.

230 Borland, pp.51-54.

231 Kernighan, p.22.

232 Microsoft, QuickBASIC reference Manual, pp.176.

233 Microsoft, Microsoft FORTRAN Compiler Reference Manual, pp.164-173.

234 Borland, pp.139-141.

235 Traister, Robert J. Going From BASIC to C (Englewood Cliffs, New jersey: Prentice Hall, Inc., 1985), p.3.

236 Cugini, p.16.

237 Ghezzi, p.54.

238 Ghezzi, p.54.

239 Ghezzi, p.54.

240 Ghezzi, p.54.

241 Kernighan, p.90-91.

242 Borland, p.119.

243 Borland, p.119.

244 Brown, p.102.

245 Brown, p.102.

246 Brown, p.103.

247 Marcotty, p.398.

248 Marcotty, p.398.

249 Kernighan, p.89.

250 Kernighan, p.89.

251 Webster's New Collegiate Dictionary, p.22.

252 United States Department of Defense, "Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A-1983", in Programming Languages: A Grand Tour, 2nd ed., Ellis Horowitz, ed. (Rockville, MD: Computer Science Press, Inc., 1985), pp.417-751, p.495.

253 Ghezzi, p.98.

254 Marcotty, p.245.

255 Marcotty, p.245.

256 Ghezzi, p.99.

257 Ghezzi, p.99.

258 Ghezzi, p.99.

259 Cugini, p.21.

260 Marcotty, p.245.

261 Cugini, p.23.

262 Marcotty, p.245.

263 Ghezzi, p.115.

264 Marcotty, p.245.

265 Ghezzi, p.115.

266 Ghezzi, p.115.

267 Marcotty, p.245.

268 Marcotty, p.245.

269 Marcotty, p.245.

270 Marcotty, p.245.

271 Marcotty, pp.245-246.

272 Marcotty, p.245.

273 Marcotty, p.245.

274 Marcotty, p.245.

275 Marcotty, p.245.

276 Marcotty, p.246.

277 Marcotty, p.246.

278 Marcotty, p.246.

279 Marcotty, p.246.

280 Cugini, p.23.

281 Marcotty, p.246.

282 Marcotty, p.246.

283 Lee, Personal Interview, 4/30/87.

284 Marcotty, p.246.

285 Marcotty, p.247.

286 Marcotty, p.247.

287 Marcotty, p.247.

288 Marcotty, p.247.

289 Marcotty, p.247.

290 Marcotty, p.247.

291 Lee, Personal Interview, 4/30/87.

292 Marcotty, p.248.

293 Marcotty, p.247.

294 Marcotty, p.247.

295 Marcotty, p.247.

296 Marcotty, p.248.

297 Kernighan, p.195.

298 Borland, pp.75-77.

299 Microsoft, Microsoft FORTRAN Compiler Reference Manual, pp.71-72.

300 Microsoft, QuickBASIC reference Manual, pp.168-169.

301 Traister, p.136.

302 Kernighan, p.93.

303 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.71.

304 Merchant, p.184.

305 Microsoft, QuickBASIC reference Manual, pp.251.

306 Borland, p.75.

307 Borland, p.75.

308 Cugini, p.23.

309 Borland, p.75.

310 Cugini, p.23.

311 Borland, pp.90-91.

312 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.69.

313 Microsoft, QuickBASIC reference Manual, pp.238-239.

314 Borland, p.76.

315 Borland, p.219.

316 Kernighan, p.104.

317 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.72.

318 Microsoft, QuickBASIC reference Manual, p.91.

319 Borland, p.79.

320 Borland, p.79.

321 MacLennan, p.200.

322 Marcotty, p.248.

323 MacLennan, p.200.

324 Marcotty, p.249.

325 Marcotty, p.249-250.

326 Marcotty, p.250.

327 Marcotty, p.250.

328 Marcotty, p.250.

329 MacLennan, p.202.

330 MacLennan, p.202.

331 MacLennan, p.202.

332 Marcotty, p.251.

333 Marcotty, p.251.

334 Lee, Personal Interview, 4/30/87.

335 Marcotty, p.251.

336 Marcotty, p.252.

337 Marcotty, p.252.

338 Marcotty, p.252.

339 Marcotty, p.252.

340 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.20.

341 Microsoft, QuickBASIC reference Manual, pp.161-162.

342 Cugini, p.27.

343 Borland, pp.79-83.

344 Borland, p.80.

345 MacLennan, p.201.

346 Cugini, p.25.

347 Lee, Personal Interview, 4/30/87.

348 Lee, Personal Interview, 4/30/87.

349 Marcotty, p.284.

350 American National Standards Committee X3, p.121.

351 United States Department of Defense, p.646.

352 Cugini, p.25.

353 Marcotty, p.284.

354 Marcotty, p.284.

355 Marcotty, p.284.

356 Kernighan, pp.151-157.

357 Borland, p.93.

358 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.134.

359 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.134.

360 American National Standards Committee X3, p.39.

361 United States Department of Defense, p.647.

362 American National Standards Committee X3, p.111.

363 Marcotty, p.284.

364 Microsoft, QuickBASIC reference Manual, pp.284,304.

365 Marcotty, p.284.

366 Marcotty, p.284.

367 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.134.

368 Marcotty, p.285.

369 Microsoft, QuickBASIC reference Manual, pp.284,304.

370 Borland, p.93.

371 Marcotty, p.285.

372 Cugini, p.25.

373 United States Department of Defense, p.645.

374 Cugini, p.25.

375 Cugini, p.25.

376 Cugini, p.27.

377 Cugini, p.27.

378 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.133.

379 Borland, p.95.

380 Microsoft, QuickBASIC reference Manual, p.304.

381 Microsoft, QuickBASIC reference Manual, p.368.

382 Borland, pp.93,101.

383 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.134.

384 Cugini, p.27.

385 Cugini, p.27.

386 Cugini, p.27.

387 American National Standards Committee X3, p.122.

388 Cugini, p.29.

389 Marcotty, p.260.

390 Cugini, p.29.

391 MacLennan, p.192.

392 MacLennan, p.192.

393 Maclennan, p.192.

394 Borland, p.87.

395 Borland, p.87.

396 Maclennan, p.192.

397 Borland, p.87.

398 Borland, pp.85-88.

399 Microsoft, QuickBASIC reference Manual, pp.161-162.

400 Microsoft, Microsoft FORTRAN Compiler Reference Manual, p.20.

401 Borland, p.87.

402 Cugini, p.29.

ADDITIONAL LANGUAGE FEATURES

6.0 Introduction

There are a variety of other factors that affect the overall abilities of a language to perform satisfactorily in certain applications. Many of these factors are unrelated to each other, yet they are important enough to briefly discuss. Each of these features is not a standard language option. They are implementation dependent, and are, therefore, difficult to discuss in exact terms. The following sections will discuss application facilities, program implementation control, simplicity, standardization, performance, software availability, software development support, mathematical support, chaining, overlays, and multi-language usage.

6.1 Application Facilities

"Application facilities characterize the major semantic functions of a language beyond those associated with particular data-types" [1]. Since these facilities are associated with the "...external entities upon which the program operates..." [2], they are more visible to the end-user instead of the programmer, who usually utilizes "...the internal features of the language" [3].

The programmer is considered to be "[a] person who designs, writes, and tests computer programs" [4]. User is defined as "one that uses" [5] where use is defined as "the method or manner of employing or applying something" [6]. Thus, end-user is considered to be a person who uses the final version of the software towards a particular purpose.

Application facilities include databases, concurrency, graphics, and libraries. [7].

6.1.1 Database

A database is defined as "a set of data, part or the whole of another set of data, and consisting of at least one file, that is sufficient for a given purpose or for a given data processing system" [8]. A program that is designed to manipulate a database is part of a database system. "Database systems are becoming more and more popular. This is due to the widespread availability of such systems on low-cost personal computers..." [9].

A language should have the ability to interface with a database system. Many databases contain much more information than an average person could find on his/her own through other sources. While interfaces to database facilities are available (in FORTRAN especially), no generalized standard currently exists for the interface between languages and database systems [10].

An interface is defined as "[a] shared boundary. An interface might be a hardware component to link two devices or it might be a portion of storage or registers accessed by two or more computer programs" [11]. Furthermore, a data transmission interface is defined as "[a] shared boundary defined by functional characteristics, common physical interconnection characteristics, signal characteristics, and other characteristics as appropriate. The concept involves the specification of the connection of two devices having different functions" [12]. A data transmission interface is essential if a program is to access the database for information.

Work has been done to standardize the interfaces (generally called bindings [13]). "ANS committee X3H2 has primary responsibility for database standards within the U.S." X3H2 has completed its work on database standards [14,15]. Thus, bindings do now exist in all standard languages [16].

None of the languages under consideration explicitly reference a database.

6.1.2 Concurrency

Concurrent is defined as "pertaining to the occurrence of two or more activities within a given interval of time" [17]. Concurrency requires communication.

> "In a system of related tasks, there must be some form of communication. For instance, we clearly do not want the trains on a rail network to collide, we may want to ensure that two bank tellers do not make conflicting transactions on the same account, or we may need to coordinate the actions of the devices in a computing system" [18].

Communication (also know as data communication) is defined as "the transmission and reception of data" [19]. "Communication is the ability for one process to pass information to another process which may be executing asynchronously [(concurrently)]. One process is a sender and the other process is a receiver of a message" [20].

"Communication occurs between two tasks whenever:

1) A send statement in one task specifies a value to be transmitted to another task, and
2) A receive statement in the other task specifies a target variable whose value is to obtained from the sending task" [21].

Two tasks meet at a rendezvous when these two conditions are met [22]. "Consider the statement

send NEW_CODE to DECODE;

which occurs in the task body for RECEIVE_CODES, and the following clause

receive CODE from RECEIVE_CODES;

taken from the body DECODE" [23]. The rendezvous is achieved when both of these statements are satisfied [24]. Thus, either RECEIVE_CODES must wait for DECODE to receive CODE (i.e., execute the 'receive CODE' statement) or DECODE must wait for RECEIVE_CODES to send NEW_CODE (i.e., execute the 'send NEW_CODE' statement) [25].

A rendezvous accomplishes two basic functions. It synchronizes the two tasks [26]. The sending task executes the send statement and must wait for the receiving task to execute the corresponding receive statement [27]. At that time, the two tasks are synchronized. The other basic function is the transmission of data [28]. A rendezvous is only one of many types of the mechanisms that can be used [29].

Type agreement between the variables of the two tasks must be maintained. "The compiler cannot detect this kind of error since the correspondence between send and receive statements cannot be determined statically" [30].

Communication between tasks is a very important concept for real-time languages. "A real-time language is one that allows the programming of procedures that can be executed concurrently and can be activated in response to external signals as required" [31]. This thesis does not consider real time languages.

None of the language implementations under consideration support communication as defined (although there are concurrent versions of these languages do exist.)

6.1.3 Graphics

"Graphics is the capability of manipulating visual objects and properties, such as locations, lines,...,and colors" [32]. Graphics is use to display a representation of a two or three dimensional object, according to the mathematical rules of projection, onto a CRT (Cathode Ray Tube) [33] or plotter [34].

Graphics was standardized in the United States in 1984 [35] by ANS committee X3H2 in working close cooperation with ISO/TC97/SC5/WG2 for international standardization [36]. QuickBASIC is the farthest advanced in terms of graphical abilities available with the standard implementation [37]. While standard Pascal does not allow graphics per se, Turbo Pascal does have good graphics abilities, especially if one uses Turbo Graphics Toolbox (Under this circumstance, Turbo Pascal surpasses QuickBASIC) [38,39]. Microsoft FORTRAN has no graphics worth mentioning (i.e., no graphics in reference manual). C has no graphics as a standard, but is mandated to have it soon [40]. However, C and FORTRAN can obtain graphics through the use of libraries that contain these routines. Thus, if the library is available, so is graphics.

6.1.4 Libraries

Webster's defines a library as "a collection resembling or suggesting a place in which literary, musical, artistic, or reference materials are kept for use but not for sale" [41]. The Accredited Standards Committee X3 defines a library as "a file or set of related files" [42]. A library, for the purposes of this thesis, is a file that contains "...already programmed, debugged, and compiled subprograms" [43].

"Needless to say, the presence of a good library can greatly simplify the programming process..." [44]. Programs utilize subprograms in these libraries by using external references [45]. Each of these subprograms may also utilize other library subprograms through external references [46]. All of the subprograms are pooled together during the linking process (See Subprograms, Chapter 4) [47]. The final result is a file that contains all of the parts of the program, including the external references [48].

The advantage of a library is that it can contain those modules that are frequently used. These modules are sometimes called general purpose modules. There are several advantageous of general purpose modules.

General purpose modules save programming time [49]. A programmer can often find a general purpose module before he/she can write a new module himself/herself [50]. However, while most programmer accept this, they rarely follow it [51]. Programmers usually feel as if they can write a subprogram in a few minutes, as opposed to spending a few hours finding the routine in a set of libraries [52] (this assumes a vast collection of libraries). In reality, it may take an hour to design the routine, a couple of hours to code it, and possibly much longer to debug it [53]. By the time the programmer's version of the routine is ready to use, it is possible that a few days have passed by [54]. Thus, if a library exists, a programmer should be familiar with it. He/She should take the time to search the available libraries for the general purpose module (for this reason, search systems are needed [55]). A library saves time by storing general purpose modules for any programmer to use.

General purpose modules save memory space [56]. If there is a team of programmers working on a computer system, each individual programmer writing his/her own special purpose modules, "...it is likely that there will be a large number of similar subroutines" [57]. In addition, if all of the independent routines were running at the same time, "there would be a great deal of wasted memory space" [58]. If a general routine were stored in a library, each programmer could access that routine, thereby eliminating the multiplicity of similar modules (this is usually handled by a configuration management system [59]). Thus, a library saves memory space by storing this module for all the programmers to access.

Using the general purpose routines aids in reducing program bugs [60]. The general purpose routine has been tested and there is a high confidence level as to its correctness. The programmer who designed the routine should have done a thorough job to ensure its correctness [61] (although this is not always true [62]). In addition, if a problem arises, a library allows easy access to the source code [63]. However, this must be done in such a way as to minimize the effect that the changes may cause to the existing programs using that subprogram. Thus, a library aids in reducing the program bugs by storing these bug free general purpose modules.

There are a couple of disadvantages to library routines. It may be difficult to locate a library routine [64]. It can be time consuming to search through vast libraries to find an appropriate routine to use [65]. This problem is usually caused by lack of documentation [66]. The general routines are usually documented poorly, if at all (in terms of explaining exactly what the routine needs, does and accomplishes) [67] (This could be solved by having standard documentation procedures). The other disadvantage is that the general routine may not conform to the programming conventions being used [68]. "Arguments may be passed to subroutines in a different way; ...; documentation style and content may be unacceptable, even though correct" [69].

Turbo Pascal is the only language under consideration that does not support libraries [70,71,72].

6.2 Program Implementation Control

"Some of the languages have features which enable the programmer to control the compilation and execution process, so as to tailor a program to a given machine, or to some optimization goal" [73]. Many of these features concern optimization of the program in terms of code size and execution speed. These features are usually implementation dependent [74]. In addition, these features usually produce a non-portable code [75]. This can be a high price to pay, depending on the intended application of the program.

QuickBASIC has a few options that are available for code optimization. These include choice of column major or row major storage, minimization of string data, speed minimization, and code size minimization. The programmer can choose to store arrays in column major order (commonly used by other BASIC implementations) or in row major order (used by many assembly language routines) [76]. The compiler can minimize amount of memory that string data (e.g., those used in input and output operations) occupies by having the program reference the same memory location for identical string literals instead of a referencing a separate location for each identical string literal [77]. The programmer can compile the program for speed, thereby decreasing the overall execution speed, but increasing the code size [78]. If the programmer chooses to minimize the storage space of the program, the execution speed may increase [79].

Microsoft FORTRAN [80] and Turbo Pascal [81] have no apparent options that affect the optimization of the code, but do have options that effect execution of the code.

Microsoft C does have options that affect code optimization. These include minimization of code size [82], minimization of execution time [83], and maximum optimization [84].

6.3 Simplicity

"A language should be easy to master. All different features should be easy to learn and remember, and the effect of any combination of features should be predictable and easily understood" [85]. "A language should be as simple as possible. There should be a minimum number of concepts with simple rules for their combination" [86].

> It is often claimed that the programmer who cannot (and does not need to) understand the entire language can live happily with a subset. As Hoare (1973) points out, this claim is not justified. Knowledge of a subset may be sufficient for programs that work as the programmer intended; but if the program does no work properly and accidentally invokes some unknown feature of the language, then serious troubles arise for the programmer [87].

In general, simplicity results in easier learning and fewer bugs [88].

Simplicity can be maintained if the language does not offer more than one way to specify a concept [89]. If more than one way is supplied, the language is larger than necessary, and a variety of different forms of the language (i.e., different language dialects) may flourish [90]. Under these circumstances, a user that understands one dialect may not be able to utilize another dialect of the same language due to the different methods of forming the concepts [91]. A typical example involves the language C. A variable may be incremented by any of the following methods:

```
1) a++
2) a=a+1
3) a+=1
4) ++a [92].
```

Only one of these methods should be used so that simplicity can be maintained.

A language may also be impaired by allowing "...different semantic concepts to be expressed by the same syntactic notation" [93]. FORTRAN can illustrate this point with the following statement:

```
SUM(I,J)=I+J [94].
```

This statement can represent two things. It can be a form of a statement function that defines the function SUM and returns the sum of the parameters I and J [95]. It could also be an assignment to the array SUM [96]. "Thus, the meaning of this statement can change radically if the user makes the common error of forgetting to declare the array SUM. Such an error is especially possible in FORTRAN because variables are not required to be declared" [97].

BASIC is the simplest of the languages [98] (QuickBASIC is the simplest of the languages under consideration). There need only be two data types within a program, variable-length strings, and numerics [99]. I/O operations can be performed without formatting [100,101]. The one statement per line convention reduces the complexity and errors commonly associated with statement separators and terminators [102].

FORTRAN and C allow straightforward and short programs to be written [103]. The syntactic overhead is higher than BASIC [104].

Pascal is more complex [105]. Its simplicity is due to its regular structure [106]. Its declarations and its rules determining statement groupings can lead to syntactic errors [107].

6.4 Standardization

Standardization has already been briefly discussed in Chapter 2. However, a brief discussion may further aid understanding its importance.

Program portability is inhibited if the language it is written in has a variety of implementations. One of the best ways to avoid this problem is to have "...precise language definitions and [to] ensure that all implementations are faithful to the definition" [108]. However, this is usually impractical. A language designer usually publishes the definition or creates an implementation of the language for general use [109]. "Once the language achieves widespread use and enough of a following [it can] become a candidate for standardization" [110].

Several organizations produce language standards. These include The American National Standards Institute (ANSI), International Organization for Standardization (ISO), and the Institute of Electrical and Electronics Engineers (IEEE).

History has shown that the language would not be standardized until it "...had existed for some time and the negative consequences of the lack of a standard became severe" [111]. However, some languages are able to achieve a standard early in their existence. There are disadvantageous to early standardization, as well as late standardization.

Early standardization will not allow the potential problems of a language to be exposed [112]. Thus, these problems would become a language standard, and all implementations would have to incorporate them [113]. "Because standards are slow - sometimes even impossible - to change, these problems would ... become permanent features of the language" [114].

However, if a language undergoes standardization late in its existence, the standard should incorporate as many of the implementation standards as possible. "It is almost impossible (and even unwise) to have a standard that will cover all these implementations" [115].

Of the languages being considered, FORTRAN has the fewest implementations deviating from the standards [116]. Since Pascal has recently (1983) been standardized, there are implementations that have not yet conformed to the standards [117]. BASIC was standardized in 1985 [118], but still suffers from the 'Street BASIC' versions. However, work is being done to standardize a newer version. Although C has not been standardized, "...the existence of a de facto standard ...[Kernighan]...has helped forestall wide divergence in implementation, but there are some differences, especially in the run-time library" [119].

6.5 Performance

"Strictly speaking, performance is a characteristic of a given implementation of a language, not of a language as such" [120]. However, certain languages encourage certain types of implementation [121].

"...[L]anguage design usually presents the problem of trading off a large number of features against compiler size and speed" [122]. These features tend to consume machine resources. "Resources include both storage and time and may be consumed during various phases of processing, such as interpretation, compilation, and execution" [123].

Performance concerns the use of resources as well as the efficiency of the language. However, the efficiency of a language involves many factors. A brief discussion is needed to help clarify the issue of efficiency.

6.5.1 Efficiency

Over the past few years, "[t]he issue of efficiency has changed considerably..." [124]. Execution speed and storage space used are not the only things that efficiency is concerned with [125]. Two additional factors have come into the efficiency realm [126]. These include the initial effort required to produce a program and the effort required in maintaining that program [127]. Thus, "...we are more concerned with the productivity of software development than the performance of the resulting products" [128]. A language supports productivity, and, therefore, efficiency, "...if it has the qualities of writeability, maintainability, and optimizability" [129].

Writeability is the property of the language that allows a programmer to be concerned with the methods of problem solving, and not be distracted by details and tricks of the language (e.g., assembly languages need for addressing mechanisms) [130]. Writeability is also the property of the programmers ability to write a program so that it is self-documenting (i.e., only a modest need for additional comments) [131].

Language maintainability incorporates two additional factors, readability and modifiability [132]. Readability concerns the process of following "the logic of the program and ... discover[ing] the presence of errors by examining the program" [133]. In one sense, the simpler the language is and the easier it is to express ideas, the easier it will be to determine what a program does from the actual code [134]. However, the simpler the language is, the more complex the program [135] is due to the possible complex algorithms needed to express complex concepts. Modifiability is primarily achieved through the modularization of a program and the use of pre-defined constants [136]. Thus, maintainability is the ease of readability of a program and the ease in which that program can be modified.

"Optimizability refers to [the] quality of allowing automatic program optimization" [137]. "Optimization refers to the process of rearranging and changing the program being compiled to produce a more efficient object program" [138]. Most of the time spent in programming is in the optimization of a code [139]. However, optimization should not be done until the program runs correctly. "The ideal approach would be first to produce a program that is demonstrably correct and then, through a series of efficiency-improving transformations, to modify this program to obtain a correct and efficient one. A language is optimizable if it makes possible the automatic application of these transformations" [140].

Now that efficiency is better understood, the languages can be characterized on performance. C, FORTRAN, and Pascal require few resources [141] and were designed to maintain efficiency [142]. Therefore, these languages are considered to be good in performance. BASIC is usually accompanied by an interpreter, which is usually slow in execution speed [143]. However, if a compiler is available (as is the case with QuickBASIC), it would similar to the FORTRAN compiler in terms of performance [144].

6.6 Software Availability

It is easier, and usually more economical, to purchase a program rather than to develop one. If the program is purchased, it is desirable to have it written in a language that is supported by an in house compiler so that modifications can be made to it if necessary to overcome compatibility problems. In addition, this program should be written in the same language as the ones it must interact with [145].

C accompanies UNIX. Almost all systems running on UNIX are developed in C [146]. Thus, there are many system routines available that can be accessed via the library mechanism on UNIX [147].

Since FORTRAN is the prevailing language for mathematical and scientific applications, there are extensive packages and libraries available in those areas [148] (e.g., IMSL, ESSL, Harwell Library, Easy Pack, Fun Pack, LINPACK, etc. [149]). FORTRAN is the language that is typically exclusively used in mathematical software [150].

BASIC dominates the world of microcomputers [151]. While BASIC is used most commonly in business software, mathematical applications are becoming more commonplace. However, since standardization of BASIC has been slow, its machine dependent versions may hinder its progress [152].

Pascal has a much smaller base in the application areas due to the other languages' dominance [153]. However, some application software is available on the microcomputer [154].

6.7 Software Development Support

The availability of tools that aid a programmer in the development and maintenance of programs is another major concern [155]. Software support can be implemented as a part of the compiler or as a separate package [156]. Software support depends strongly upon the vendor and changes over time [157].

In general, BASIC has editing and debugging support in many of its implementations [158]. FORTRAN has "...a wide variety of commercially available software support, although this is very often vendor dependent" [159]. While software support for C and Pascal is less extensive than the other languages, particular implementations may offer extra support [160].

Since the availability of software aids is not considered to be part of the language [161], the following sections briefly discuss some of the aids commonly considered to be important.

6.7.1 Source Code Manipulation and Checking

Source code manipulation and checking concerns all facilities that aid the development and modification of source code [162]. A language's ability to incorporate pre-written sections (i.e., through an INCLUDE statement) of code can significantly aid the development process. All of the languages under consideration allow this to occur [163,164,165,166].

"External to the languages are such features as editors, prettyprinters, and library managers" [167]. A prettyprinter is software that will print out a program in a pretty format (e.g., using indentation, page breaks, titles on each page, page numbers, etc.). A library manager is software that helps "...keep track of such matters as which versions of code have been compiled, which modules depend on others, etc." [168]. Language-based editors allow the programmer to write the program and have easy access to the compiler. "Such editors incorporate many of the syntax rules of the language and thus are capable of automatically generating syntactically correct code in response to user commands" [169].

Software is available that will check code to ensure adherence to language standards and conventions [170]. Some of these will allow the user to dictate his/her own conventions to be enforced [171]. Others will check against the ANSI standard [172]. This software is usually built into the compilation process [173].

6.7.2 Program Testing

Program testing aids are software used to detect syntax and logic errors [174]. They do not produce correct code [175]. These aids are available in language embedded, compiler embedded, or external forms [176].

The DEBUG statement is supported by various languages. Microsoft FORTRAN [177] and QuickBASIC [178,179] (as well as BASIC) both support versions of this. C supports a concept called conditional compilation. This allows the compiler to ignore certain sections of code based on various conditions [180,181]. Commenting out sections of code can also be used as a debugging practice. C and Pascal allow the user to comment out large sections of code easily by adding a '(*' to begin and a '*)' to end. FORTRAN and QuickBASIC require considerably more work by requiring a comment signifier to be placed on every line of code.

"Compiler and run-time diagnostics can be of great value when testing code" [182]. "Especially for those languages without exception handling, it is very important that the system provide useful information when encountering anomalous run-time conditions" [183]. Since these diagnostics should be helpful to identify problems, they should be clear and easy to understand [184].

6.7.3 Information and Analysis

Software is also available that can "...analyze both the logical and performance characteristics of programs" [185]. Such facilities include "...cross reference tables for identifiers, statistics on the uses of different types of statements, object code listings, and statistics on storage and time usage..." [186]. It is difficult to point out exactly which vendors offer these facilities, although many of them are supported in compilers. However, FORTRAN usually supports many of these [187].

6.8 Mathematical Support

Mathematical support refers to the languages ability to incorporate mathematical libraries and use of a personal computer's math coprocessor (8087 or 80287). Using the math coprocessor can increase accuracy and speed of computations.

Microsoft FORTRAN [188] offers mathematical library support as well as math co-processor support. QuickBASIC does not directly have additional math libraries [189] nor does it currently directly support the math coprocessor [190]. However, via an overlay from another vendor, it is possible to have both of these capabilities [191]. Turbo Pascal does not have options for math coprocessor support; a whole different compiler supplied by Turbo must be used to access the math coprocessor [192]. Microsoft C supports both math libraries and the mathematical coprocessor [193].

6.9 Chaining

Chaining is the process of linking programs together. A chain command "transfers control from [the] current program to another program" [194]. Execution of the second program begins at the first statement of the code and continues in normal program execution. Control is not transferred back to the original program (unless specifically arranged through another chaining command). Data may be transferred between the programs via some form of a COMMON statement [195].

QuickBASIC is the only language under consideration that allows chaining to occur [196] (See Figure 6.1).

6.10 Overlays

Overlays optimize computer memory. An "...overlay system lets [a programmer] create programs much larger than can be accommodated by the computer's memory" [197]. The general idea is to collect subprograms in one or more files that are not part of the main program [198]. These files will "...be loaded automatically one at a time into the same area in memory" [199].

Figure 6.2 shows a schematic of how this is done. The main program is resident in memory. One of the overlay files, which contains five subprograms, is resident on disk.

Figure 6.3 demonstrates how one of the subprograms is used by the main program. When one of the subprograms is called, it is loaded into the reserved overlay area, at the first address of that overlay area [200], in the main program [201] . This reserved area must be large enough to accommodate the largest of the subprograms [202]. The space that the main program requires is reduced by the space occupied by the largest of these subprograms [203]. Figure 6.3 shows Subprogram 2 loaded into the main program. Since Subprogram 2 is the largest of the subprograms in the overlay file, it occupies all of the overlay area.

Figure 6.4 shows Subprogram 3 loaded at the first address of the overlay area. Since Subprogram 3 is smaller that Subprogram 2, it will not use all of the overlay area [204]. The remainder of the overlay area is unused [205].

When one of these subprograms is loaded, it may not reference one of the other subprograms [206]. The other subprograms are not in memory, and, therefore, do not exist. Attempting to reference an unloaded subprogram will produce an error.

It is possible to create more than one overlay area in the main program (See Figure 6.5) [207]. At this time, a second file would be needed to provide the subprogram for that area (i.e., one subprogram file per overlay area) [208]. Since the subprograms do not occupy the same memory area [209], a subprogram in one overlay area may call a subprogram in the other overlay area [210]. Figure 6.5 shows that Subprogram 3 can call Subprogram 1 or Subprogram 2 [211]. In addition, Subprogram 1 and Subprogram 2 may call Subprogram 3 [212]. However, Subprogram 1 and Subprogram 2 may not call each other [213].

The only disadvantage of an overlay system is the increased disk access time required to load each subprogram [214]. However, if an overlay system is designed well, this increased time is nearly negligible [215].

The actual process of creating overlays is usually implementation dependent. However, these implementations usually automatically create the overlay areas as well as the overlay files [216]. Thus, the programmer merely uses a special command to denote overlay procedures, while everything else is done automatically.

Turbo Pascal is the only language under consideration that allows the use of overlays [217] due to its 64K limitation on program size [218].

6.11 Multi-Language Usage

Multi-language usage a method whereby several different languages can be used to compose a program. This process tends to combine the effects of the languages (e.g., BASIC's graphics, FORTRAN's math and science base, C and Pascal bit manipulation). This method is vendor dependent.

For example, the input and output routines may be written in BASIC (for its screen control), while the calculation routine may be written in FORTRAN (for its broad math base). Other languages could be combined for various reasons. Thus, it would be advantageous if a program could interact with routines written in other languages.

QuickBASIC will allow interaction with assembly language [219]. These routines are subprograms stored in a library and accessed by a QuickBASIC program via a CALL statement [220]. Since it is a subprogram, parameter passing is allowed [221].

Microsoft FORTRAN will allow interaction with any language that produces linkable object modules [222]. Thus, it does not matter which language, but that the object module can be linked to the FORTRAN program. Object files commonly come from Microsoft FORTRAN, Microsoft Pascal, user codes in other high level languages, and assembly language routines [223].

Turbo Pascal will not allow any interaction with other languages.

Microsoft C allows the use of other languages. These include assembly language, Microsoft FORTRAN, and Microsoft Pascal [224].

6.12 Conclusion

A database can greatly aid a person in finding information that he/she may need. Information is typically stored in a database (i.e., a libraries computerized card catalog and most other information retrieval systems). Thus, a programmer may need to design a program that manipulates a database. For this reason, programmers should be familiar with database properties. While an engineer may not need to design this type of a program, his familiarity with database concepts may increase the speed in which he/she can use a database to his/her advantage.

Communication between two concurrently running programs is a very important property, especially on larger computer systems. The proper communication between operating programs that need data from each other is essential. A communication rendezvous accomplishes two main tasks: 1) the synchronization of programs, and 2) the transmission of data. While concurrent applications are not yet commonly used on personal computers, they are frequently used on mainframes. Most engineers will not need to program in concurrent languages. However, if an engineer is considering this option, he/she must be familiar with the communication process.

Graphics is an important issue for many engineering applications. In computer languages used in engineering, graphics must be supported. An engineer can make extensive use of graphics in pre and post processors. Therefore, the languages should support extensive graphics abilities. QuickBASIC is the only language under consideration that supports extended graphics in its standard. Turbo Pascal supports some graphics. Microsoft FORTRAN and Microsoft C do not support graphics at all, yet may implement graphics routines from libraries.

A language should support libraries. Libraries are files that contain subprograms. A good library can greatly increase the efficiency of a programmer. Good libraries save programming time, save memory space, and reduce program bugs. Libraries enhance the modularity concepts of programming. With extensive libraries, it is possible to design a program that implements the already designed, debugged, and efficient subprograms that are held in these libraries. Even though libraries are usually implementation dependent, all of the languages under consideration, except Turbo Pascal, allow the use of libraries.

Program implementation control is primarily concerned with the optimization of program code (usually in terms of speed and storage). However, extreme caution must be used with this feature since it may make a program non-portable. Code optimization can be done automatically during the compilation process. QuickBASIC and Microsoft C allow the optimization of program code. However, Microsoft FORTRAN and Turbo Pascal have no apparent optimizing facilities, but do have options that effect code execution. Since optimization of code decreases execution time as well as storage space, it should be supported.

Languages should be as simple as possible. A language's features should be predictable, easy to understand, and easy to master. The simplicity of a language can determine the ease with which it can be learned and mastered. The simpler a language is, the easier it is to use. Simplicity is best maintained by allowing concepts to be expressed in only one way (as opposed to having several ways to form the same concept). However, if languages are too simple, concepts may not be able to be adequately developed. QuickBASIC is the simplest of the languages being considered. Microsoft FORTRAN and C follow QuickBASIC, with Pascal being the most complex.

Standardization of a language increases the portability of that language. In addition, standardization will increase the number of programmers that can successfully code from implementation to implementation since all implementations must conform to that standard. Of the languages being considered, FORTRAN has been standardized the longest, with Pascal being recently standardized. A full structured form of BASIC was standardized in 1985, but still suffers form the 'Street BASIC' dialects. Although C is not yet standardized (a standard is expected soon), its different implementations do not vary much.

A language should be good in performance. Performance concerns the language's use of resources as well as the language's efficiency. Efficiency no longer concerns just speed of execution and minimal storage space used. Efficiency is now measured in terms of program development. Efficiency in program development is measured in terms of writeability, readability, optimizability, maintainability, and modifiability. C, FORTRAN, and Pascal all use few resources and are efficient, and, therefore, are considered to be good in terms of performance as long as the algorithms being implemented are good. QuickBASIC is also considered to be good in performance.

When developing a program, one should consider the option of buying software to perform the desired task. It may be cheaper and more efficient to purchase this software rather than developing the software in house. If the software is to be purchased, it should be written in the same language that the other programs being used are written so that compatibility problems may be avoided and the program may be easily modified if necessary. Of the languages being considered, FORTRAN and BASIC are more available than Pascal and C.

Software development support concerns the development and maintenance of programs. FORTRAN and BASIC tend to have a lot of development support, while C and Pascal lag behind in this area. A language should support some form of source code checking. All of the languages under consideration allow this to occur. Software available for information and analysis is commonly built into the compiler of the languages.

Mathematical support is critical in engineering applications. Any language used in engineering should support mathematical libraries and the mathematical coprocessor. Turbo Pascal, C, and FORTRAN directly support the mathematical coprocessor. While QuickBASIC does not directly support the mathematical coprocessor, it is possible to obtain a mathematical package to accompany QuickBASIC that does. Microsoft C and Microsoft FORTRAN support math libraries. Turbo Pascal and QuickBASIC do not.

Chaining is not considered a good programming practice since the only method of transferring data is through some form of a COMMON statement. QuickBASIC is the only language under consideration that allows the chaining of programs.

An overlay system allows the programmer to create a program that is much larger that can be maintained in computer memory at one time. An overlay system makes efficient use of available memory. A language should support the use of overlays. Turbo Pascal is the only language under consideration that supports overlays.

There are times when it is necessary to use more than one language in an application. This process can maximize the benefits of a variety of languages. If this is to be done, languages need to be able to support the codes of other languages. Microsoft FORTRAN support Microsoft C and Microsoft Pascal. Similarly, Microsoft C supports Microsoft FORTRAN and Microsoft Pascal. QuickBASIC supports only assembly language. Turbo Pascal does not support other languages.

```
' This program is MAIN.BAS:
COMMON A(1),N$,B,LN
  INPUT; "Base (enter 0 to end): ",B
  IF B=0 THEN
     END
  ELSE
     INPUT "                    Number: ",N$
  END IF
  LN=LEN(N$)
  DIM A(LN)
  CHAIN "DIGIT"

'This program is DIGIT.BAS:
COMMON A(1),N$,B,LN
FOR I=1 TO LN
    DGT$=MID$(N$,I,1)
    IF DGT$>="0" AND DGT$<="9" THEN
        A(I)=VAL(DGT$)
    ELSEIF DGT$>="A" AND DGT$<="Z" THEN
        A(I)=ASC(DGT$) - ASC("A") + 10
    ELSEIF DGT$>="a" AND DGT$<="z" THEN
        A(I)=ASC(DGT$) - ASC("a") + 10
    ELSE
        PRINT "Illegal input value."
        CHAIN "MAIN"
    END IF
NEXT
CHAIN "DEC"

'This program is DEC.BAS:
COMMON A(1),N$,B,LN
  DECIMAL=0
  FOR I=1 TO LN
      DECIMAL= B*DECIMAL + A(I)
  NEXT
  ERASE A
  PRINT "Decimal # = " DECIMAL : PRINT
CHAIN "MAIN"
```

Figure adapted from: Microsoft, Microsoft QuickBASIC Compiler (Redmond, Washington: Microsoft Incorporated, 1986), p.211.

FIGURE 6.1 - THE CHAIN COMMAND IN QUICKBASIC

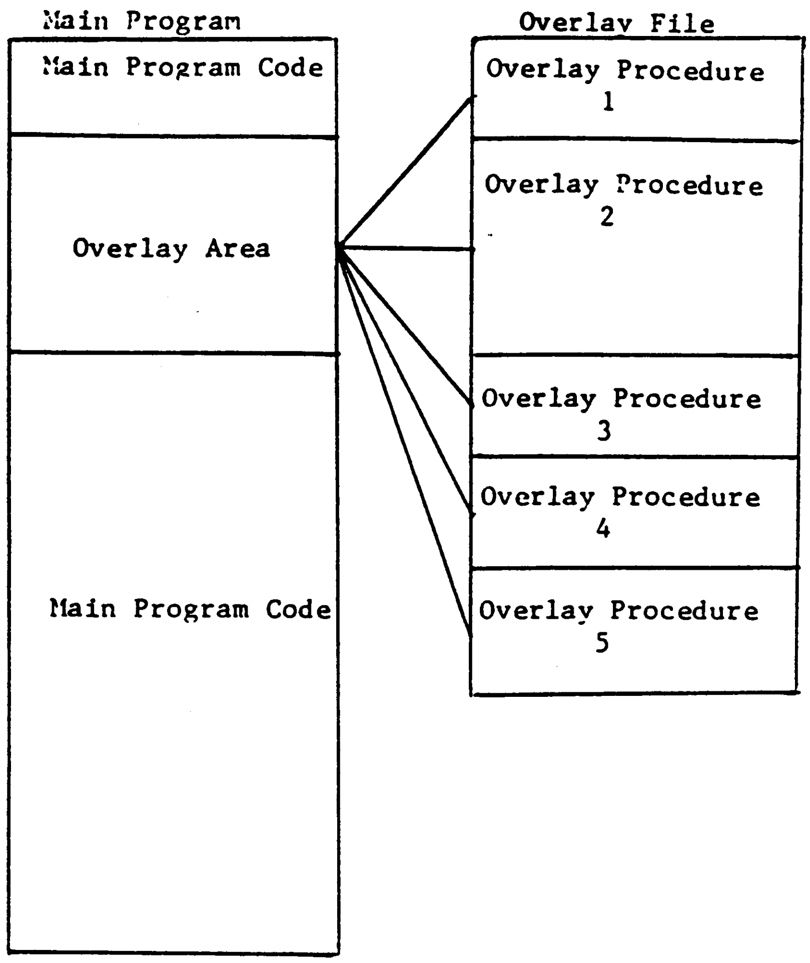


Figure adapted from: Borland, Turbo Pascal (Scotts Valley, California: Borland International Inc., 1986), p.149.

FIGURE 6.2 - PRINCIPLE OF OVERLAY SYSTEM

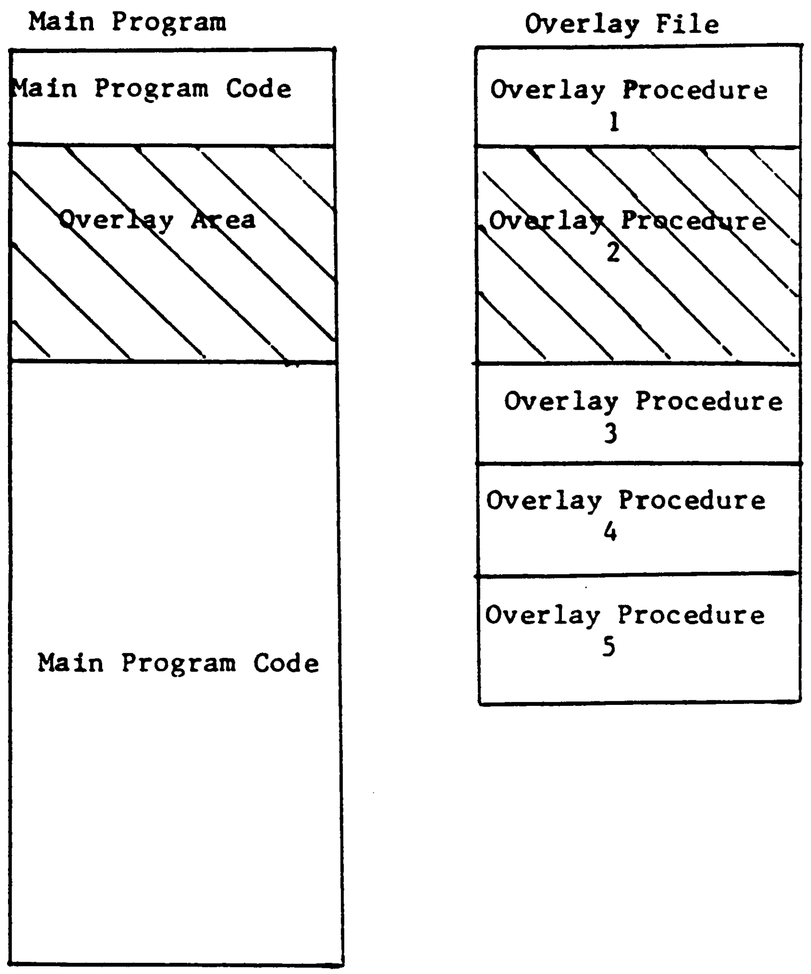


Figure adapted from: Borland, Turbo Pascal (Scotts Valley, California: Borland International Inc., 1986), p.150.

FIGURE 6.3 - LARGEST OVERLAY SUBPROGRAM LOADED

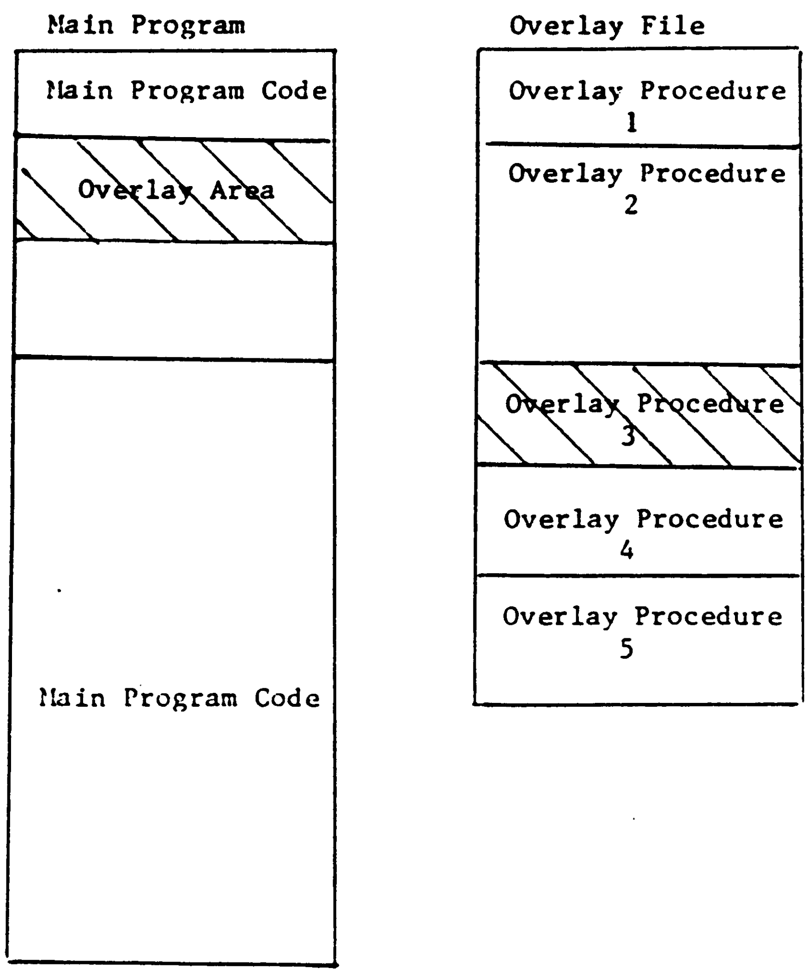


Figure adapted from: Borland, Turbo Pascal (Scotts Valley, California: Borland International Inc., 1986), p.151.

FIGURE 6.4 - SMALLER OVERLAY SUBPROGRAM LOADED

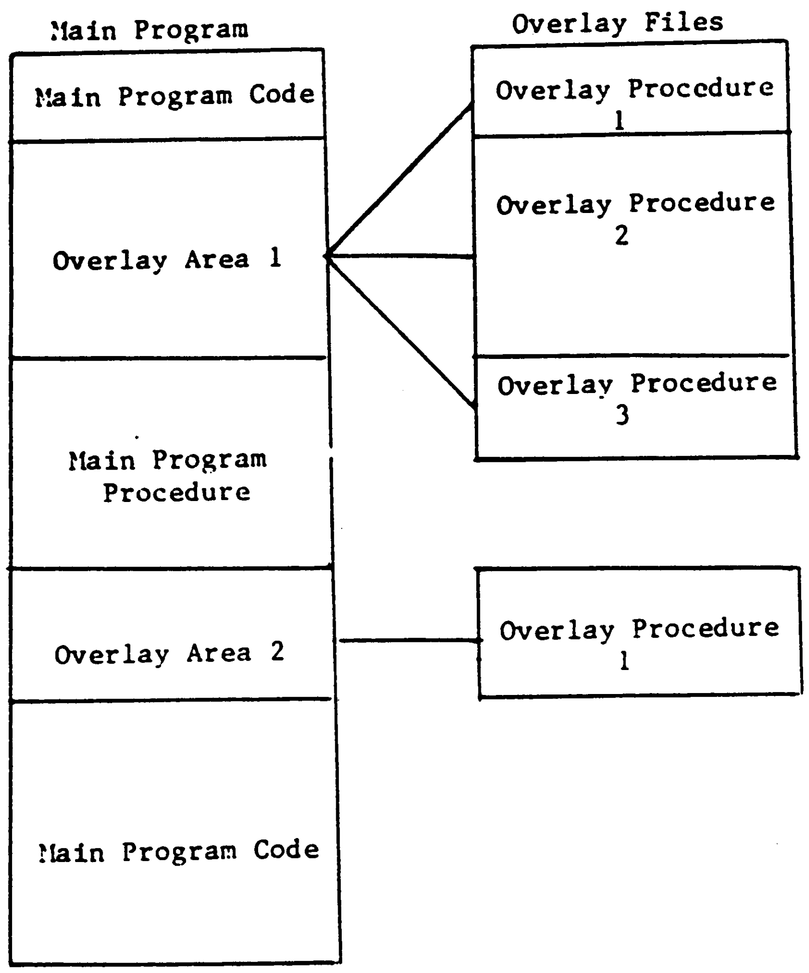


Figure adapted from: Borland, Turbo Pascal (Scotts Valley, California: Borland International Inc., 1986), p.153.

FIGURE 6.5 - MULTIPLE OVERLAY FILES

NOTES - CHAPTER 6

1 Cugini, p.29.

2 Cugini, p.29.

3 Cugini, p.29.

4 American National Standards Committee X3, p.105.

5 _Webster's New Collegiate Dictionary_, p.1279.

6 _Webster's New Collegiate Dictionary_, p.1279.

7 Cugini, p.vi-vii.

8 American National Standards Committee X3, p.33.

9 Ghezzi, p.294.

10 Cugini, p.32.

11 American National Standards Committee X3, pp.67-68.

12 American National Standards Committee X3, p.36.

13 Lee, Personal Interview, 5/1/87.

14 Cugini, p.32.

15 Lee, Personal Interview, 5/1/87.

16 Lee, Personal Interview, 5/1/87.

17 American National Standards Committee X3, p.28.

18 Marcotty, p.466.

19 American National Standards Committee X3, p.33.

20 Cugini, p.32.

21 Marcotty, p.466.

22 Marcotty, p.467.

23 Marcotty, p.467.

24 Marcotty, p.467.

25 Marcotty, p.467.

26 Marcotty, p.467.

27 Marcotty, p.467.

28 Marcotty, p.467.

29 Lee, Personal Interview, 5/1/87.

30 Marcotty, p.467.

31 Marcotty, p.7.

32 Cugini, p.33.

33 Webster's New Collegiate Dictionary, p.497.

34 Lee, Personal Interview, 4/15/87.

35 Lee, Personal Interview, 5/1/87.

36 Cugini, p.33.

37 Microsoft, QuickBASIC Reference Manual, pp.215, 222, 253, 286, 325, 360, 379, 383, 389, 391, 400, 404, 415, 418, 445, 446, 504, 513.

38 Borland, pp.159-185.

39 Borland International, Turbo Graphics Toolbox (Scotts Valley, Ca: Borland International Inc., 1985), pp.1-156.

40 Lee, Personal Interview, 5/1/87.

41 Webster's New Collegiate Dictionary, p.656.

42 American National Standards Committee X3, p.73.

43 MacLennan, p.39.

44 MacLennan, p.39.

45 MacLennan, p.39.

46 MacLennan, p.39.

47 MacLennan, p.39.

48 MacLennan, p.39.

49 Yourdon, Techniques of Program, p.126.

50 Yourdon, Techniques of Program, p.126.

51 Yourdon, Techniques of Program, p.126.

52 Yourdon, Techniques of Program, p.126.

53 Yourdon, Techniques of Program, p.126.

54 Yourdon, Techniques of Program, p.126.

55 Lee, Personal Interview, 5/1/87.

56 Yourdon, Techniques of Program, p.126.

57 Yourdon, Techniques of Program, p.126.

58 Yourdon, Techniques of Program, p.127.

59 Lee, Personal Interview, 5/1/87.

60 Yourdon, Techniques of Program, p.127.

61 Yourdon, Techniques of Program, p.127.

62 Lee, Personal Interview, 5/1/87.

63 Yourdon, Techniques of Program, p.127.

64 Yourdon, Techniques of Program, p.128.

65 Yourdon, Techniques of Program, p.128.

66 Yourdon, Techniques of Program, p.128.

67 Yourdon, Techniques of Program, p.128.

68 Yourdon, Techniques of Program, p.128.

69 Yourdon, Techniques of Program, p.128.

70 Microsoft, QuickBASIC Reference Manual, p.133.

71 Microsoft, Microsoft FORTRAN Compiler Library Reference Manual (Redmond, Washington: Microsoft Corporation, 1986), p.21.

72 Microsoft, Microsoft C Compiler Users Guide (Redmond, Washington: Microsoft Corporation, 1986), p.150.

73 Cugini, p.33.

74 Cugini, p.33.

75 Cugini, p.33.

76 Microsoft, QuickBASIC Reference Manual, p.91.

77 Microsoft, QuickBASIC Reference Manual, p.91.

78 Microsoft, QuickBASIC Reference Manual, p.93.

79 Microsoft, QuickBASIC Reference Manual, p.94.

80 Microsoft, Microsoft FORTRAN Compiler Reference Manual, pp.177-196.

81 Borland, pp.313-315.

82 Microsoft Corporation, Microsoft C Compiler User's Guide (Redmond, Washington: Microsoft Corporation, 1986.), p.91.

83 Microsoft, Microsoft C Compiler User's Guide, p.91.

84 Microsoft, Microsoft C Compiler User's Guide, p.203.

85 Ghezzi, p.335.

86 MacLennan, p.527.

87 Ghezzi, p.335.

88 Cugini, p.34.

89 Ghezzi, p.335.

90 Ghezzi, p.335.

91 Ghezzi, p.335.

92 Ghezzi, p.336.

93 Ghezzi, p.336.

94 Ghezzi, p.337.

95 Ghezzi, p.337.

96 Ghezzi, p.337.

97 Ghezzi, p.337.

98 Cugini, p.34.

99 Cugini, p.34.

100 Cugini, p.34.

101 Microsoft, QuickBASIC Reference Manual, pp.406,301,304.

102 Cugini, p.34.

103 Cugini, p.34.

104 Cugini, p.34.

105 Cugini, p.35.

106 Cugini, p.35.

107 Cugini, p.35.

108 Ghezzi, p.90.

109 Ghezzi, p.90.

110 Ghezzi, p.90.

111 Ghezzi, p.90.

112 Ghezzi, p.91.

113 Ghezzi, p.91.

114 Ghezzi, p.91.

115 Ghezzi, p.91.

116 Cugini, p.35.

117 Cugini, p.35.

118 Lee, Personal Interview, 5/5/87.

119 Cugini, p.37.

120 Cugini, p.37.

121 Cugini, p.37.

122 Cugini, p.37.

123 Cugini, p.38.

124 Ghezzi, p.10.

125 Ghezzi, p.10.

126 Ghezzi, p.10.

127 Ghezzi, p.10.

128 Ghezzi, p.10.

129 Ghezzi, p.10.

130 Ghezzi, p.8.

131 Marcotty, p.13.

132 Ghezzi, p.9.

133 Ghezzi, p.9.

134 Ghezzi, p.9.

135 Lee, Personal Interview, 5/5/87.

136 Ghezzi, p.9.

137 Ghezzi, p.10.

138 Marcotty, p.137.

139 Ghezzi, p.10.

140 Ghezzi, pp.10-11.

141 Cugini, p.38.

142 Cugini, p.38.

143 Cugini, p.38.

144 Cugini, p.38.

145 Cugini, p.38.

146 Cugini, p.39.

147 Cugini, p.39.

148 Cugini, p.39.

149 Kirstein, Dean, User's Services Consultant, Personal Interview, 4/20/87.

150 Cugini, p.39.

151 Cugini, p.39.

152 Cugini, p.39.

153 Cugini, p.39.

154 Cugini, p.39.

155 Cugini, p.39.

156 Cugini, p.39.

157 Cugini, p.39.

158 Cugini, p.40.

159 Cugini, p.40.

160 Cugini, p.40.

161 Cugini, p.39.

162 Cugini, p.40.

163 Microsoft, Microsoft FORTRAN Reference Manual, p.185.

164 Borland, Turbo Pascal, p.147.

165 Microsoft, QuickBASIC reference Manual, p.188.

166 Kernighan, p.86.

167 Cugini, p.40.

168 Cugini, p.40.

169 Cugini, p.40.

170 Cugini, p.41.

171 Cugini, p.41.

172 Cugini, p.41.

173 Cugini, p.41.

174 Cugini, p.41.

175 Cugini, p.41.

176 Cugini, p.41.

177 Microsoft, Microsoft FORTRAN Reference Manual, p.181.

178 Microsoft, QuickBASIC reference Manual, p.90.

179 Microsoft, QuickBASIC reference Manual, p.61.

180 Kernighan, p.208.

181 Cugini, p.41.

182 Cugini, p.41.

183 Cugini, p.41.

184 Cugini, p.41.

185 Cugini, p.42.

186 Cugini, p.42.

187 Cugini, p.42.

188 Microsoft, Microsoft FORTRAN Users Guide (Redmond, Washington: Microsoft Corporation, 1985), p.58.

189 Microsoft, QuickBASIC reference Manual, pp.535-545.

190 Microsoft Corporation, phone call.

191 Microsoft Corporation, phone call.

192 Borland, Turbo Pascal, pp.301-302.

193 Microsoft, Microsoft C Compiler User's Guide, pp.39,79-84.

194 Microsoft, QuickBASIC reference Manual, p.209.

195 Microsoft, QuickBASIC reference Manual, p.209.

196 Microsoft, QuickBASIC reference Manual, p.209.

197 Borland, p.149.

198 Borland, p.149.

199 Borland, p.149.

200 Borland, p.151.

201 Borland, p.150.

202 Borland, p.150.

203 Borland, p.150.

204 Borland, p.151.

205 Borland, p.151.

206 Borland, p.151.

207 Borland, p.153.

208 Borland, pp.152-153.

209 Borland, p.156.

210 Borland, p.156.

211 Borland, p.156.

212 Borland, p.156.

213 Borland, p.156.

214 Borland, p.151.

215 Borland, p.151.

216 Borland, pp.152-153.

217 Borland, pp.149-157.

218 Holzer, S.M., Personal Interview, 4/27/87.

219 Microsoft, QuickBASIC reference Manual, p.139.

220 Microsoft, QuickBASIC reference Manual, p.139.

221 Microsoft, QuickBASIC reference Manual, pp.144-146.

222 Microsoft, Microsoft FORTRAN Users Guide, p.55.

223 Microsoft, Microsoft FORTRAN Users Guide, p.55.

224 Microsoft, Microsoft C Compiler User's Guide (Redmond, Washington: Microsoft Corporation, 1986), p.229.

APPLICATION REQUIREMENTS

7.0 Introduction

Once the individual languages have been analyzed, criteria need to be established for application requirements. The topics discussed are short and are intended as a guideline to aid in the choice of languages. Application requirements include functional operations, size and complexity, expertise, end-user interaction, reliability, timeframe, portability [1], and memory limitations [2].

7.1 Functional Operations

The language must be able to perform the task at hand. Applications can be broadly classified into two categories: business oriented (fixed point arithmetic, character manipulation, and file handling) and scientifically oriented (array manipulation and numerical calculation) [3].

Data types and application facilities are only two of the many things that can affect this category. However, a programmer should determine the full scope of wLat is needed and not rely solely upon two or three categories [4]. "When no one language appears adequate [for] the task, users should consider a multi-language approach - although there are associated costs, especially because of the lack of standardization of inter-language capabilities" [5].

BASIC and QuickBASIC offer the best support for the business oriented (information processing) programs [6]. However, due to their lack of support of a record construct, they may be somewhat difficult to implement in certain situations (i.e., to form lists) [7].

For scientific, mathematical and engineering applications, FORTRAN offers the strongest support, especially when one considers the software available for purchase [8]. BASIC (as well as QuickBASIC) should also be considered due to its large library of built-in functions, its extended precision floating-point numbers, and its array manipulation [9].

Since C allows fairly direct access to the actual machine, it is suitable for systems programming [10].

BASIC (as well as QuickBASIC) and Pascal are suitable for educational application since they "...allow clear and simple expressions for the concepts being taught" [11]. BASIC and QuickBASIC teach the simple concepts of computers and programming. Pascal teaches basic programming theory [12].

7.2 Size and Complexity

The size and complexity of the programs being developed should be a major concern. The size refers to the number of lines in the code. Complexity can be defined as the amount of interaction between modules and/or the sophistication of the algorithms [13].

"For large, complex applications, language features which support the creation and maintenance of code (software availability and development support) or help the programmer to express complex relationships (program structure) are especially relevant" [14]. However, none of the languages under study are truly suitable for large and complex applications. However, FORTRAN is commonly used in the finite element analysis areas to produce large and relatively complex programs.

Small programs (<250 lines) require that large syntactic and conceptual overhead be minimal and that simplicity be a crucial property [15]. BASIC (as well as QuickBASIC) and FORTRAN are well suited for such applications [16].

Pascal and C are suitable for applications between these two areas [17].

7.3 Expertise

The language used can greatly effect the development of software. This can be best explained by the following example. An engineering firm needs to write a program to accomplish some goal. If the language used is unfamiliar to the staff, either the staff will have to be trained, or the program will have to be contracted out [18]. It is under this circumstance that a comparison of short-term costs and long-term gains must be considered [19]. If programming is to be an important part of the firm, then training the staff can be considered to be a worthwhile investment. However, if this is the only program that will be developed by the firm, then contracting it out is probably a better investment.

If the programming is to be done in-house, then the application to be programmed may be done by a professional programmer or by one who does programming as a secondary skill [20]. Many engineering firms do not have professional programmers on staff. "With the spread of microcomputers, casual programming is becoming more common" [21]. Casual programming (i.e., programming by people who consider programming to be a secondary skill) should be done in a language that is simple and offers many application facilities to the programmer [22]. BASIC (and QuickBASIC) and FORTRAN are suitable for casual programming [23].

Professional programmers can use complex languages [24]. These languages have a variety of features that would enable a professional programmer to "...make good use of more sophisticated features" [25]. However, none of the languages under consideration are considered complex languages.

Pascal and C are suitable for intermediate cases [26]. While C is a relatively simple language, "...programming in C involves the construction and use of sophisticated libraries of system functions from C's simple repertoire of features" [27].

The language that the programmers know is an important factor if the firm is considering hiring these programmers [28]. Standardization of a language plays a key role in this determination [29] since each programmer should know the standard version. FORTRAN, BASIC (not QuickBASIC), and Pascal are the languages that are most commonly known [30].

7.4 End User Interaction

Interactive processing is that feature that allows the user to interact with the software through input/output (I/O) operations. "Interactive processing...requires features to support that type of I/O operation" [31]. In addition, "...run-time performance [must] be adequate for quick response to user actions" [32].

BASIC is the strongest language available for interactive applications due to its strong I/O and graphics support [33]. Pascal is weak in interactive processing. The other languages perform reasonably well [34].

7.5 Reliability

Many applications perform tasks that demand a high degree of reliability [35]. "Program structure features, especially exception-handling, help meet this requirement. Type checking is also thought to be useful in early detection of certain kinds of errors" [36]. In addition, "software development support and language simplicity ... contribute to the construction of correct programs" [37]. The languages themselves must support the ability to write reliable programs. "It is not the languages that are reliable, but the programs written in them" [38].

BASIC and QuickBASIC have few data types and allow exception handling, so they is suitably support reliability [39]. "Pascal offers strong type checking, but no exception handling..." [40], and is therefore considered to acceptably support reliability.

Structured programming techniques are very important in creating a reliable program. If structured programming techniques are used extensively, a high degree of reliability will automatically be incorporated into the program, no matter what language is used.

7.6 Timeframe

> Some programs are written, executed a few times, and then discarded. Others become part of a system which may be operational for decades. Clearly, ease of writing is a dominant concern in the first case, and ease of reading in the second [41].

Features that effect the timeframe of the program's existence are program structure, software availability, and software development [42]. Standardization is an important issue to be considered when a long-term program is being developed [43], "...since it is very possible that the system will outlive the hardware on which [the system] is originally implemented" [44].

BASIC, FORTRAN, and possibly C are appropriate for short-term programs since "...their emphasis is on simplicity and low syntactic overhead" [45] (ease of writing). Pascal is appropriate for long-term programs since its "...emphasis is on more detailed declarations and elaborate compilation" [46] (ease of reading).

7.7 Portability

Portability is the ability to re-use source code on different systems [47]. Portability is best achieved through strict coding practices of the user [48]. Portability cannot be guaranteed by the language alone [49].

Standardization is the key criterion for this requirement [50]. Thus FORTRAN and Pascal are highly suitable for this purpose [51]. C is also suitable since it is relatively invariant between installations [52].

7.8 Memory Limitations

The amount of memory that must be accessed to accomplish the task can be a very important criterion, especially in engineering applications. Many languages have memory limitations when operating on a personal computer. Thus, the programmer should know the memory limitations of a language implementation before language selection.

Turbo Pascal is limited to a 64K code segment size and a 64K data segment size [53]. This can be overcome by the use of the overlays [54] and pointers [55] that Turbo Pascal allows the user to utilize, although this process increases the complexity of the program.

The compiler in QuickBASIC will compile only up to a 64K code size [56]. However, the linker will link as many of the units as memory will allow [57]. QuickBASIC is limited to 64K in data segment size for strings and single variables [58]. Each array is stored in a far heap and is limited to 64K in size, but there may be as many arrays as memory allows [59].

Microsoft FORTRAN will only compile a 64K code size at a time [60]. The linker will produce up to one megabyte of executable code [61]. Microsoft FORTRAN is limited to 640K of memory for all of its data [62]. Microsoft C has the same limitations as Microsoft FORTRAN [63].

7.9 Conclusions

A programmer must determine the operations that a program needs to perform before choosing a language to program in. He/She should consider a wide scope of factors and not limit himself/herself to just a couple of options. The language chosen should support as many of the operations as is possible. It may also be necessary to consider a multi-language program if all of the desired functions cannot be obtained through one language. For example, one language could be used for the pre-processor, another for the processor, and still another for the post-processor.

The size and complexity of the program needs to be considered. FORTRAN is used for large complex programs. FORTRAN and QuickBASIC are appropriate for small implementations. Pascal and C are suitable for intermediate programs.

The expertise of the programmers abilities must be considered. Short-term cost vs. long-term gains must be considered. If programming is to be done in-house then a few additional factors should be considered. If programming is a secondary skill for the programmer, then a simple language, such as QuickBASIC or FORTRAN, needs to be used due to their low syntactic overhead. However, caution must be taken since these languages also tend to encourage errors due to their implicit declarations, weak type checking, and lack of good block structure. If programming is the primary skill (i.e., a professional programmer), then a complex language can be used (i.e., Ada, PL/1, COBOL). Pascal and C can be used for those programmers who are between these areas. QuickBASIC and FORTRAN will be usable for any ability level of the programmers.

The degree to which the end-user is to interact with the program should be determined. Certain languages allow more interaction than others. QuickBASIC is the best language under consideration for this purpose for its ease of I/O operations and ease of manipulating the screen. Pascal is the weakest. The other languages fall between these two.

Structured programming techniques are the best approach to a reliable program. However, exception handling, type checking, software development support and language simplicity are a few additional factors that may aid in producing a reliable program. QuickBASIC, for its simplicity and exception handling, and Pascal, for its type checking, are the best languages under consideration for this purpose.

The length of time that the program will exist can be another factor to consider. QuickBASIC, FORTRAN, and C are languages that are easy to write and that allow quick development. These languages are appropriate for short-term programs. Pascal encourages lengthy declarations, and therefore, increased readability. Pascal is suitable for long-term programs. While these are the ideal situations, reality demonstrates that FORTRAN is used for long term programming as well.

Portability is best achieved through strict programming practices of the user and through standardization. Of the languages being considered, FORTRAN and Pascal are the strongest in terms of standardization.

Any form of memory limitation will decrease the effectiveness of a language. QuickBASIC and Turbo Pascal have limitations on the amount of data that may be processed. Microsoft

FORTRAN and Microsoft C have no limitations on data. All of the languages under consideration restrict the source code size to 64K. However, Microsoft QuickBASIC, Microsoft FORTRAN, and Microsoft C overcome this limitation by allowing the linker to link as many of these 64K code segments as memory will allow. Turbo Pascal implements an overlay system to overcome this limitation.

NOTES - CHAPTER 7

1 Cugini, p.42-49.

2 Foutz, Personal Interview.

3 Cugini, p.42.

4 Cugini, p.42.

5 Cugini, p.42.

6 Cugini, p.45.

7 Cugini, p.45.

8 Cugini, p.45.

9 Cugini, p.45.

10 Cugini, p.45.

11 Cugini, p.45.

12 Cugini, p.45.

13 Cugini, p.45.

14 Cugini, p.45.

15 Cugini, p.46.

16 Cugini, p.46.

17 Cugini, p.46.

18 Cugini, p.46.

19 Cugini, p.46.

20 Cugini, p.46.

21 Cugini, p.46.

22 Cugini, p.46.

23 Cugini, p.46.

24 Cugini, p.46.

25 Cugini, p.46.

26 Cugini, p.46.

27 Cugini, p.46.

28 Cugini, p.47.

29 Cugini, p.47.

30 Cugini, p.47.

31 Cugini, p.47.

32 Cugini, p.47.

33 Cugini, p.47.

34 Cugini, p.47.

35 Cugini, p.47.

36 Cugini, p.47.

37 Cugini, p.47.

38 Lee, Personal Interview, 5/13/87.

39 Cugini, p.47.

40 Cugini, p.47.

41 Cugini, p.48.

42 Cugini, p.48.

43 Cugini, p.48.

44 Cugini, p.48.

45 Cugini, p.48.

46 Cugini, p.48.

47 Cugini, p.48.

48 Cugini, p.48.

49 Cugini, p.48.

50 Cugini, p.48.

51 Cugini, p.48.

52 Cugini, p.48.

53 Borland, p.226.

54 Borland, pp.149-157.

55 Borland, p.362.

56 Microsoft, phone call 4/29/87.

57 Microsoft, phone call 4/29/87.

58 Microsoft, phone call 4/29/87.

59 Microsoft, phone call 4/29/87.

60 Microsoft, phone call 4/29/87.

61 Microsoft, phone call 4/29/87.

62 Microsoft, phone call 4/29/87.

63 Microsoft, phone call 4/29/87.

LANGUAGE PERFORMANCE

8.0 Introduction

Once the languages have been evaluated on structured programming, syntactic style, semantic structure, and data type and manipulation their processors should be compared on speed and correctness. These categories are difficult to exactly evaluate. There are a variety of test methods available to determine the speed, efficiency, and accuracy of a given program. The results will vary from implementation to implementation as well as from computer to computer.

8.1 Methods of Evaluation

There are three basic methods, among many others, that can be used to evaluate a program. Each one measures a different aspect of the program.

Experiments can be performed to determine the ease of developing an algorithm and debugging a program using the various languages [1]. The general idea is to give a number of programmers specific specifications for the intended application [2]. The program must be developed in each of the considered languages. The programmers may decide in which language they wish to program. The programmers must work independently. They would carefully monitor the time spent in algorithm development, coding, and debugging [3]. Furthermore, they would record the number of bugs found, and the time spent correcting each bug [4]. Thus, an evaluation on the ease of program development would be obtained in each language.

There are software engineering procedures available that measure the complexity of a program [5]. The program can be developed in each language. The software engineering procedures could then be used to determine the complexity of each version of the program [6]. Thus, a general idea of the complexity of each language could be obtained to aid in determining the ease of program coding.

The performance of each language can also be measured. This is usually done using various 'benchmark' tests. "In any benchmark comparison it is important that conditions be exactly duplicated for each product being compared. In cases where conditions cannot be exactly duplicated, they should be as similar as possible" [7].

Each of the methods described above has distinct advantages. However, from the practical standpoint, only one can be used in this thesis. The group development project requires a large number of programmers and a large amount of time. The time needed was not available. The programmers were not available. The software engineering procedures are complex and would take a considerable amount of time to learn. Again, the time was not available. Thus, the performance of the languages using a benchmark test was the method chosen for this thesis.

8.2 Program Performance

Performance was measured by using the matrix solve routine used in Holzer's recently modified Plane Frame Analysis Program. The solve routine is based on Klaus-Jurgen Bathe's 'Skyline Reduction Method' used in the COLSOL (COLumn SOLver) subroutine [8,9] of his program STAP (Structural Analysis Program) [10].

"Program COLSOL is an active column solver to obtain the LDL^T factorization of a stiffness matrix [and/]or reduce and back-substitute the load vector. The complete process gives the solution of the finite element equilibrium equations" [11]. Examples and explanations of the process are available in Bathe [12] and Holzer [13].

Since only the solve routine was used, a driver was created to input the data required by the routine. This data was obtained from a modified version of Holzer's Plane Frame Analysis Program. Test cases were designed and executed on the Va. Tech mainframe VM1 system. The number of elements (NE), number of joints (NJ), number of load conditions (NLC), member code matrix (MCODE), system stiffness matrix (SS) and the force vector (Q) prior to the SOLVE routine, and the displacement vector (Q) (used for comparative study) after the SOLVE routine were all results of Holzer's Plane Frame Analysis Program. This data was downloaded to an AT&T 6300A personal computer.

The driver performs several functions. It calls procedures to input and echo the required data (NE, NJ, NEQ, NLC, MCODE, SS, the force vector Q. It also calls the SKYLIN routine.

The SKYLIN procedure uses the MCODE to form the column height vector KHT. KHT is used to form MAXA, which stores the addresses of the diagonal elements of the system stiffness matrix. SKYLIN also determines the length of the system stiffness matrix (LSS).

After all of the required data is obtained the SOLVE routine is called. On the first load condition, SOLVE calls FACTOR. FACTOR performs the LDL^T factorization of the system stiffness matrix using MAXA and SS. FACTOR is called only on the first load condition. After FACTOR, and for all subsequent load conditions, REDUCE and BACSUB are called. REDUCE reduces the right-side load vector using MAXA, SS, and Q. BACSUB performs Back-Substitution to obtain the solution using MAXA, SS, and Q. Upon completion of BACSUB, the displacement vector, Q, is printed out.

Performance is measured by measuring the execution time and the correctness of the various SOLVE routines. Execution time was determined by printing the time to the screen and output file immediately before and after SOLVE. The difference in times is calculated. However, Microsoft FORTRAN has no method for accessing the clock. Thus, 'BEGIN' and 'END' are printed and the time was determined manually. Turbo Pascal has no feature for determining the difference in time, thus the difference is determined manually. This process was repeated ten times for the versions accessing the 8087 math coprocessor and for the versions not accessing the 8087 math coprocessor in each of the considered languages. In addition, the total compilation and linking time (from source code to executable code) was recorded for reference (See Figure 8.3). The results of this process can be seen in Figure 8.1. Source code listings of the programs used can be found in Appendices A through D.

The test case structure used to obtain the execution times in Figure 8.1 is presented in Figure 8.2. This is a frame with sixteen floors and four bays. The structure has 80 free joints (the base joints are fixed) and 144 elements, which yields 240 degrees of freedom (D.O.F.). Each element has the following properties:

Area	5 in^2
Moment of Inertia	50 in^4
Modulus of Elasticity	30000 ksi
Coefficient of Thermal Expansion	0.0000065/^{o}F.

The first load condition is a 0.2 k/in distributed load on each floor. The second load condition is a horizontal force of 1.5^k at each free joint. Due to the length of the output, only the QuickBASIC version accessing the 8087 math coprocessor is presented in Appendix E.

8.3 Program Correctness

The execution speed of a program is not crucial if that program produces inaccurate results [14]. There are three classic ways to demonstrate the correctness of an algorithm. The first of these involves obtaining a second solution by a different method [15]. This is a form of 'Fault Tolerance' [16]. The two solutions should yield acceptably close results.

Another method is to substitute the solution back into the original specification (e.g., form a programmer's standpoint, the actual equation may not be know, while the specifications for the program must be known) to ensure that the result meets the requirements [17]. This is a commonly used practice in many applications.

The last classical method involves mathematical proofs [18]. This can be a very intensive and exhaustive procedure. This method is not commonly used.

In the programming environment, a variation of the first approach is used. 'N-version' programming is a process through which correctness is demonstrated when 'N' people, independently developing the same program, obtain the same results [19]. The odds of the two programs being incorrect in the same way are highly improbable [20].

Correctness of the developed programs was verified by using the 'N-version' approach. The results of Holzer's Plane Frame Analysis Program were used as a basis for comparison.

A program was written in QuickBASIC to accomplish this task (See Appendix F). SKYACC inputs the results from the various programs and subtracts them from the corresponding results from Holzer's program. The difference for each equation was calculated and recorded. The average difference was also determined (See Figure 8.4).

8.4 Accuracy

The speed of program execution is not important if that program produces inaccurate results. The accuracy of numeric computations must be discussed. This topic is sometimes taken for granted when it probably should not be.

There are problems with a language standard attempting to specify numeric accuracy. One problem concerns the lack of conformity of word lengths among various hardware (i.e., a word is defined as "a character string or a binary element string that is convenient for some purpose to consider as an entity" [21] that is "stored in one computer location and [is] capable of being treated as a unit" [22]; word length is defined as "the number of characters or binary elements in a word" [23]). "More importantly, there is the general problem of characterizing the behavior...of discrete simulation of the mathematics of real numbers" [24]. "But real mathematics is inherently non-discrete" [25]. An irrational number "...leads to diverse implementation[s]" [26]. Diverse implementations of languages make it difficult to adopt "...an acceptable accuracy criterion" [27].

There are three things that cause inaccuracies:

1) Granularity: "the inability of a discrete machine to represent real numbers exactly" [28].
2) "The inherent instability of some functions and operations for certain arguments. It is easy to produce cases where a small error (relative or absolute) in the argument will generate a much larger error in the result" [29]. This can interact with granularity to produce drastic errors [30].
3) Poor algorithms, both at the hardware level for elementary operations and in subroutine libraries for functions [31].

"Clearly, when testing how well a computer system has implemented its mathematical capabilities, we need somehow to build into our test criterion a recognition of the unavoidable limitations imposed by the first two items" [32].

8.4.1 Solution Errors

Errors can occur in any computer program. The errors inherent "...in the solution of linear equations have been studied within the contexts of numerical analysis" [33]. Papers and textbooks devoted to this topic are available.

The system stiffness matrix may be *ill-conditioned.* A small change in 'K' or 'q' "...may lead to large changes of coefficients in the solution vector..." Q [34]. A small change in 'K' or 'q' may, or may not, effect 'Q' since 'Q' may not be sensitive to those changes. "Ill-conditioning may reflect the physical reality of a structure with low tangent stiffness because it is near buckling or collapse" [35]. "...An analysis can be performed which shows that it is not only a small (near zero) eigenvalue λ_1 but a large ratio of the largest to the smallest eigenvalues [the condition number] of 'K' that determines the solution errors " [36]:

$$\text{cond}(K) = \lambda_n / \lambda_1$$ [37,38,39].

Thus, a large condition number indicates that solution errors are more likely [40]. In addition, a symptom of ill-conditioning is represented by large values of K^{-1} [41].

Ill-conditioning can be caused by *truncation error* [42,43] and *rounding error* [44,45].

> Truncation error is the more important. Consider a computer that uses 'p' bits per word. Only the leading 'p' bits of a stiffness coefficient K_{ij} can be stored. It is possible...that information essential to an accurate solution resides in the trailing bits of K_{ij}. If the number is truncated, this information is discarded. The information content of [K] is therefore inadequate and cannot be restored by subsequent processing of [K], however accurately done [46].

Rounding numbers for the initial input is also considered as a truncation error [47].

The adjustment of the last bit during solution processes is denoted as rounding error [48]. "Experience and tests have shown that in finite element analysis, round-off errors are not as serious as the initial truncation errors, and indeed it appears that for an error estimate only initial errors must be considered" [49,50].

The effect of truncation errors can be estimated [51]. The number of accurate digits, 's', obtained in a solution can be estimated based on 't', the number of precise digits represented by the computer [52]:

$$s >= t - \log_{10}(\text{cond}(K))$$ [53,54,55].

Another error that may occur involves the Pivot. If $K_{pivot} <= 0$ then the structure is unstable [56]. However most of the structures tested are known to be stable. Since the structure is stable, the system stiffness matrix is guaranteed to be positive definite [57].

The Diagonal Decay Test [58,59] "...may reveal ill-conditioning during Gaussian elimination..." [60]. This test checks the diagonal coefficients of 'K'. In particular, the ratio K_{pivot}/K_{kk} is computed, where K_{pivot} is the pivot, and K_{kk} is the initial diagonal coefficient [61,62]. If this ratio becomes unacceptably small, execution should be terminated [63,64]. For a computer with a 'p' bit mantissa, a ratio of $1000/2^p$ might be acceptable [65].

An equilibrium check should be performed to determine if the solution is correct. The solution error is

$$q - \bar{q} = K^{-1} \Delta Q$$ [66]

where $\bar{q}$ is the computed solution. Thus, the residual ΔQ, which should be computed in double precision, represents the unbalanced forces [67]. If the residual is large, then there is a problem with the solution [68]. "However, a small residual does not guarantee an accurate solution because relatively large values of K^{-1}, a symptom of ill-conditioning, can cause a large solution error even for small residuals" [69].

8.5 Additional Comments

There is one last important topic that should be discussed. This concerns the availability and efficiency of the various languages.

FORTRAN is currently the only language that is extensively supported on a super computer [70,71]. In addition, "...FORTRAN has come to dominate scientific, engineering, and mathematical applications" [72,73,74]. Thus, FORTRAN is the dominant language where finite element applications are concerned [75].

FORTRAN also allows a different type of flexibility and more convenience in the use of arrays. FORTRAN is the only language under consideration that allows the programmer to reserve a large array, partition it out through the use of pointers, and pass the desired part of that array to a procedure (See Appendix A). This process allows a little more flexibility in solution processes.

8.6 Conclusion

The speed of a program is no longer considered to be a crucial factor in language implementation. The overall life cycle of the software is now considered the principle concern. Program correctness must be verified before the speed of execution is considered. Execution speed may be increased after the correctness has been verified. The opposite is not always true.

By studying Figure 8.1, the results of SOLVE execution can be evaluated. These results include the factorization of the system stiffness matrix. The results rank the languages C, FORTRAN, Pascal, and QuickBASIC (from fastest to slowest).

By studying Figure 8.4, the results of the correctness of the SOLVE routine can be determined. The comparison was based on the results of Holzer's program. All of the various programs performed reasonably well with a minimum average relative difference of $2.7*10^{-4}$. Thus, on the basis of this test, all of the languages are approximately equal in numerical accuracy.

The error involved in the solution process must be considered. Several factors lead to the error of a program. There are tests available to aid in detecting an ill-conditioned matrix. However, there is no test to guard against truncation and rounding. Truncation is the worst of these two errors. Double precision may guard against rounding, but not against truncation. Since there is not a way of preventing these errors from occurring (and they inevitably will), the solution must be checked. The diagonal decay test may be used during execution to determine if there has been a significant loss of figures. An equilibrium check should also be performed at the end of each run to ensure that the residual is minimal. While these tests cannot guarantee the accuracy of the solution, they can aid in detecting possible problems.

All times listed in seconds:
1st Load Condition/2nd Load Condition

With 8087		Without 8087	
Microsoft FORTRAN			
5.37/1.54	5.71/1/37	5.60/1.41	5.65/1.42
5.90/1.24	5.75/1.41	5.61/1.28	5.60/1.54
5.84/1.63	5.71/1.52	5.68/1.39	5.66/1.47
5.62/1.49	5.65/1.51	5.56/1.43	5.70/1.22
5.59/1.44	5.64/1.27	5.69/1.54	5.73/1.37
Average 5.68/1.44		Average 5.65/1.41	
QuickBASIC			
55.50/10.10	55.50/10.20	55.59/10.20	55.60/10.20
55.50/10.10	55.50/10.10	55.50/10.10	55.50/10.20
55.60/10.20	55.50.10.20	55.50/10.20	55.50/10.20
55.50/10.09	55.60/10.20	55.50/10.20	55.60/10.20
55.50/10.20	55.50/10.10	55.50/10.10	55.60/10.20
Average 55.52/10.16		Average 55.54/10.18	
Turbo Pascal			
13.0/2.3	12.9/2.3	32.4/5.9	32.4/5.8
13.0/2.3	12.9/2.3	32.3/5.8	32.4/5.8
12.9/2.2	12.9/2.3	32.4/5.8	32.4/5.8
12.9/2.2	12.9/2.3	32.4/5.8	32.4/5.8
12.9/2.2	13.0/2.2	32.4/5.8	32.4/5.8
Average 12.93/2.26		Average 32.39/5.81	
Microsoft C			
4/1	4/0	3/1	3/1
4/0	4/0	3/1	3/0
4/1	4/1	3/0	3/1
4/1	3/0	3/1	4/0
4/1	4/1	4/0	3/1
Average 3.9/.6		Average 3.2/.6	

FIGURE 8.1 - EXECUTION TIMES

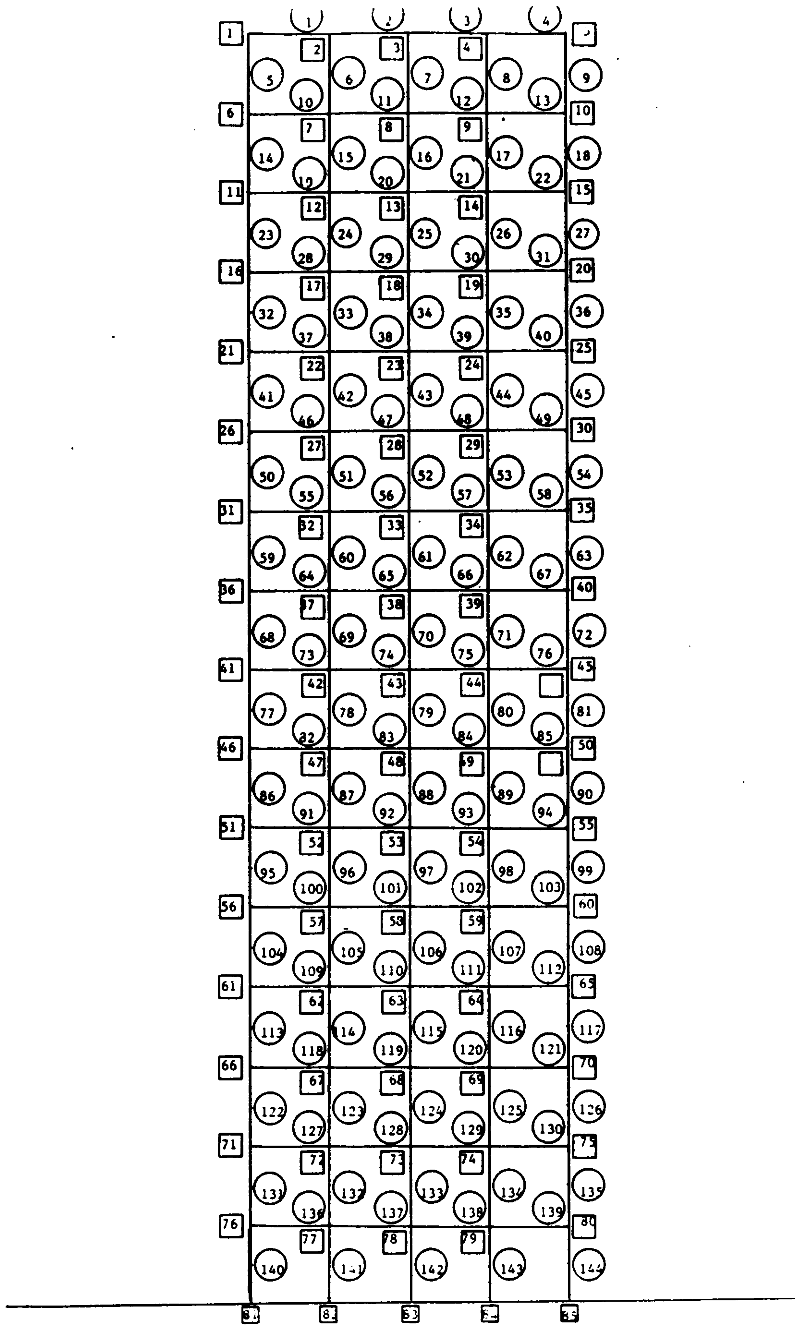


FIGURE 8.2 - TEST CASE

All times listed in minutes and seconds (xx:xx)

With 8087		Without 8087
	Microsoft FORTRAN	
2:02		2:02
	QuickBASIC	
0:20		0:20
	Turbo Pascal	
0:04		0:04
	Microsoft C	
3:00		3:02

FIGURE 8.3 - COMPILE TIMES

With 8087		Without 8087
	Microsoft FORTRAN	
$2.7124 * 10^{-5}$		No Difference
	QuickBASIC	
$2.715 * 10^{-5}$		No Difference
	Turbo Pascal	
$2.727625 * 10^{-5}$		$2.727625 * 10^{-5}$
	Microsoft C	
$2.712042 * 10^{-5}$		$2.573625 * 10^{-5}$

FIGURE 8.4 - ACCURACY OF PROGRAMS

NOTES - CHAPTER 8

1 Lee, Personal Interview 5/8/87.

2 Lee, Personal Interview 5/8/87.

3 Lee, Personal Interview 5/8/87.

4 Lee, Personal Interview 5/8/87.

5 Lee, Personal Interview 5/8/87.

6 Lee, Personal Interview 5/8/87.

7 Hartwig, Glenn. "BYTE Benchmark Tests for System and Software Reviews, Version 1.5" (Peterborough, New Hampshire: McGraw-Hill).

8 Bathe, Klaus-Jurgen, Finite Element Procedures in Engineering Analysis (Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1982), pp.448-449.

9 Bathe, pp.721-722.

10 Bathe, pp. 714-725.

11 Bathe, p.448.

12 Bathe, pp.432-449.

13 Holzer, pp.280-323.

14 Lee, Personal Interview 5/8/87.

15 Lee, Personal Interview 5/8/87.

16 Lee, Personal Interview 5/8/87.

17 Lee, Personal Interview 5/8/87.

18 Lee, Personal Interview 5/8/87.

19 Lee, Personal Interview 5/8/87.

20 Lee, Personal Interview 5/8/87.

21 American National Standards Committee X3, p.145.

22 American National Standards Committee X3, p.28.

23 American National Standards Committee X3, p.145.

24 Cugini, John V., <u>Specifications and Test Methods for Numeric Accuracy in Programming Language Standards</u>, National Bureau of Standards Special Publication 500-77 (Washington, D.C.: U.S. Government Printing Office, 1981), p.5.

25 Cugini, <u>Specifications and Test Methods</u>, p.6.

26 Cugini, <u>Specifications and Test Methods</u>, p.6.

27 Cugini, <u>Specifications and Test Methods</u>, p.6.

28 Cugini, <u>Specifications and Test Methods</u>, p.8.

29 Cugini, <u>Specifications and Test Methods</u>, p.6.

30 Cugini, <u>Specifications and Test Methods</u>, p.6.

31 Cugini, <u>Specifications and Test Methods</u>, p.6.

32 Cugini, <u>Specifications and Test Methods</u>, p.6.

33 Holzer, <u>Computer Analysis of Structures</u>, p.316.

34 Cook, Robert D., Concepts and Applications of Finite Element Analysis (New York, New York: John Wiley & Sons, 1981.), p.409.

35 Cook, p.409.

36 Bathe, p.485.

37 Bathe, p.485.

38 Cook, p.412.

39 Holzer, <u>Computer Analysis of Structures</u>, p.317.

40 Bathe, p.485.

41 Holzer, <u>Computer Analysis of Structures</u>, p.319.

42 Bathe, p.482.

43 Cook, p.410.

44 Bathe, p.482.

45 Cook, p.410.

46 Cook, p.410.

47 Bathe, p.481.

48 Cook, p.410.

49 Bathe, p.481.

50 Cook, p.410.

51 Holzer, p.318.

52 Bathe, p.486.

53 Bathe, p.486.

54 Cook, p.412.

55 Holzer, *Computer Analysis of Structures*, p.318.

56 Cook, p.415.

57 Holzer, Person Interview, 5/19/87.

58 Holzer, *Computer Analysis of Structures*, p.318.

59 Cook, p.415.

60 Holzer, *Computer Analysis of Structures*, p.318.

61 Holzer, *Computer Analysis of Structures*, p.318.

62 Cook, p.415.

63 Holzer, *Computer Analysis of Structures*, p.318.

64 Cook, p.415.

65 Cook, p.415.

66 Holzer, *Computer Analysis of Structures*, p.319.

67 Holzer, *Computer Analysis of Structures*, p.319.

68 Holzer, *Computer Analysis of Structures*, p.319.

69 Holzer, *Computer Analysis of Structures*, p.319.

70 He, Xudong, Doctor of Philosophy Candidate in Computer Science, Personal Interview, 5/12/87.

71 Abu-Sufah, Walid, Associate Professor, Va. Tech Computer Science Department, Personal Interview, 5/18/87.

72 Cugini, Selection and Use, p.54.

73 He, Xudong, Doctor of Philosophy Candidate in Computer Science, Personal Interview, 5/12/87.

74 Abu-Sufah, Personal Interview, 5/18/87.

75 He, Personal Interview, 5/12/87.

SUMMARY AND CONCLUSION

Structured design is a set of concepts that aid the decomposition of a problem into manageable subproblems by using the basic building block of the atom to produce the block structures concatenation, selection and repetition. Adhering to structured design processes, such as the top-down approach and/or the bottom-up approach, and the concepts of structured programming will produce a program that is easier to design, code, debug, modify, and, most importantly, maintain. These techniques minimize the number of bugs and the time spent in the debugging process. Data has been accumulated which shows that the time and cost of the program's design, coding, debugging, and testing is significantly decreased. The programmer's morale is increased. The increased readability and understandability of the program yields ease of maintenance and modification. Thus, an engineer can greatly increase the efficiency of program production and maintenance by the use of structured design and structured programming techniques.

Microsoft QuickBASIC, Microsoft FORTRAN, Microsoft C, an Turbo Pascal are specific implementations of languages that are commonly in use today. These languages have been reviewed on a variety of points in the preceding chapters. Tables 1 through 7, adapted from Cugini, summarize the major points of those chapters and list them in oraer of their percent of importance to engineering applications.

The syntactic style of a language can greatly increase or decrease the readability of a program as well as the ease with which the programmer can make the program readable. Table 1 lists the language features that support syntactic style in order of their percent importance.

Turbo Pascal ranks highly in each of these categories, and is, therefore, the language that best supports syntactic style. Since Microsoft C does not allow as many identifier choices, it follows Turbo Pascal. Microsoft QuickBASIC ranks ahead of Microsoft FORTRAN since QuickBASIC allows more identifier choices, free format and comments to appear at the end of a line.

Semantic structures are features that allow the programmer to represent the meaning of an algorithm or data or both. Semantic structure can be divided into the control of data and the control of execution.

The control of data refers to the ability of a programmer to control the representation of data in a program. Table 2 ranks the language features that support control of data from most important to least important by percent.

Microsoft C best supports the control of data since it allows the definition of local data inside blocks, and allows the use of external data when necessary. Although Turbo Pascal fully supports data abstraction, it follows Microsoft C since it does not allow ιhe definition of local variables inside blocks within subprograms and since data abstraction is not commonly used in engineering applications. Microsoft QuickBASIC and Microsoft FORTRAN follow Turbo

Pascal since they have weak type checking and do not use memory as efficiently because they are static languages.

The control of execution concerns the features of a language that describe the algorithmic structure of a program. Language features supporting the control of execution are listed in Table 3 in the order of their importance in terms of percentages.

Microsoft C best supports the control of execution due to its block structuring. Turbo Pascal actually has better ratings in terms of internal functions, but it must be noted that internal functions can be represented equally as well by external functions, thereby making Turbo Pascal a secondary choice due to its lack of full block structuring. Microsoft QuickBASIC is better than Microsoft FORTRAN due to its ability to handle exceptions.

Parameter Passing should also be considered. Parameter passing is summarized in Table 4. Pass by reference is preferable in engineering applications. However, pass by value is needed to pass values to functions since they automatically return a single value and should not affect the data they use. Microsoft QuickBASIC and Turbo Pascal offer the most variety by supporting both methods. Microsoft FORTRAN supports pass by reference only, while Microsoft C directly supports pass by value only. Microsoft C can emulate pass by reference through the use of pointers. Microsoft Fortran, Turbo Pascal and QuickBASIC support Pass by Result through the results of functions.

The type possessed by a variable determines what type (i.e., integer, real, etc.) of value may be associated with the variable as well as what type of operations can be performed using those values. It is important that the languages be able to support and check the various data types. The major language features that support data type and manipulation are presented in Table 5 in order the order of their importance in terms of percent.

From an engineering standpoint the following items should be noted:

1) numerical operations must be extensively supported
2) extended precision is important
3) many numeric functions should be available
4) arrays must exist (subscripts need only be integer, and dimensions past the third are not important)
5) while the record data type has many uses, it is not commonly used in engineering applications (discussed later).

With these observations and the results presented in Table 5, Microsoft FORTRAN is most desirable in this area due to its support of complex numbers, and many available functions. Microsoft QuickBASIC would be the preferred over Turbo Pascal due to its support of extended precision and many available functions. Microsoft C's lack of numeric functions, raise to the power, and direct or internal files make it undesirable.

There is a variety of factors that affects the overall abilities of a language to perform satisfactorily in certain applications. While many of these factors are unrelated, they are important enough to consider in language evaluation. Some of the available factors are presented in Table 6 in their order of importance.

Microsoft QuickBASIC best supports many of these additional factors. Of the remaining languages, Microsoft C and Microsoft FORTRAN are more desirable since their library support will allow future expansion. (It must be noted that while Turbo Pascal does not support the use of libraries, most other Pascal versions do.).

Language selection should not occur until the application requirements have been established. It is only by knowing what the application needs to do that the desirable language features can be determined. Some of these features are presented in their order of importance in Table 7. Each one of these features is subject to debate. However, based on the results of Table 7, Microsoft FORTRAN is the most desirable due to its math and science stronghold, portability, full utilization of available memory for data, and ease of use for the programmer. Microsoft QuickBASIC is the second choice due to its ability to run standard BASIC programs that have a stronghold in math and science, are reliable, are able to support the casual programmer, and support of end-user interaction. Turbo Pascal is not commonly used in the math and science fields, and is extremely limited in its data size abilities, even though this can be increased through the use of pointers (which will, in turn, increase the size and complexity of the program). Microsoft C is a new mid-level language that is not widely known and can be difficult to learn.

The performance of the languages should also be considered. Performance was measured on the speed and relative correctness of the SOLVE routine of the CE4002 Plane Frame Analysis Program. This routine uses matrix factorization and/or forward and back substitution to determine the solution to the simultaneous equations. This routine was developed in the four languages under consideration. All of the languages were reasonably close in terms of accuracy. In terms of execution speed, the ranking from fastest to slowest would be Microsoft C, Microsoft FORTRAN, Turbo Pascal, and Microsoft QuickBASIC.

There are practical considerations that should be taken into account. The availability of the language must be considered. This is an important category. This category could overpower the other findings of this thesis. Of the languages being considered, FORTRAN is the most dominant in the math, science, and engineering fields. BASIC, Pascal, and C (in order of decreasing availability) follow FORTRAN. FORTRAN is also the only language that is currently fully supported on a super computer. In terms of finite element applications, FORTRAN has been used to develop large and complex programs that have been in existence for years (i.e. SAP). These two important considerations make FORTRAN that much more desirable.

The concepts of structured programming must not be forgotten. C is the language under consideration that is most structured, followed closely by Pascal. FORTRAN and QuickBASIC are not highly favored in this area. Improvements need to be made to FORTRAN in order for it to be considered a structured language. Despite this serious drawback, FORTRAN is still widely used and accepted in today's engineering applications.

Upon review of the above conclusions, each of these languages is equally desirable (i.e., a four way tie, excluding language performance, practical considerations and structured programming). Language performance favored C and FORTRAN. The practical considerations weigh heavily in favor of FORTRAN. Structured programming concepts favor C and Pascal. Despite this, it is difficult to ignore the popularity and widespread support of FORTRAN, especially with FORTRAN (along with the other languages) standards consistently being modified to support structured programming. Thus, FORTRAN is still the language that should be used in engineering applications.

A few final comments may offer a little more insight as to why things are the way they are, and as to what can be done to correct them.

Microsoft QuickBASIC is not a standard form of BASIC. It is one form of the new structured versions of BASIC being created today (other versions are True BASIC and Better BASIC). Structured programming techniques have altered the design philosophy and criteria of many of the languages. BASIC implementations that are similar to the infamous 'Street BASIC' must be abolished. QuickBASIC is an effort towards this goal. It is hoped that it will evolve into a better language without losing the qualities that it currently has.

Since FORTRAN is the dominant language in the engineering environment, many data types and manipulation processes are not known. FORTRAN does not support the record type, data abstraction, sets, and some other types. It is for this reason that these concepts are not commonly known or used in the engineering environment. If FORTRAN had supported these concepts, they may be in common use today. It is interesting to note that the dominance of one language, such as FORTRAN, can cause this adverse situation to occur.

The dominance of FORTRAN significantly affects the results of this thesis. If FORTRAN were not such a domineering force in the computing environment, the results may have been different (i.e., Pascal or C). FORTRAN has been in existence longer than C or Pascal. C and Pascal may, with time, gain a broad base in the mathematical, science, and engineering fields. At that time, if FORTRAN has still not evolved into the structured language that it should be, C and Pascal may begin to seriously compete with FORTRAN.

FORTRAN77 has been in existence for over a decade. It is now time for a new revision of the language. FORTRAN, and therefore Microsoft FORTRAN, needs to be improved. FORTRAN must become a more structured language. This is one of the key areas in which FORTRAN77 is lacking. FORTRAN77 needs to be revised to include such structured techniques as: 1) the third basic control structure of the Bohm-Jacopini theorem (the while-do loop); 2) full support of the concept of block structuring; 3) definition of local data within blocks; 4) declaration of all variables; 5) strict type checking as a direct result of declarations; 6) limiting the number of coercions; and 7) the abolishment of the COMMON statement in favor of the record data type. While some of these changes could be easily made, others may take a little doing. However, the long term gains will easily justify the troubles and expenses of making these corrections.

TABLE 1 - SYNTACTIC STYLE

Language Features Listed in order of % importance		Microsoft QuickBASIC	Microsoft C	Microsoft FORTRAN	Turbo Pascal
Declared Entities	40%	No	Yes	No	Yes
Free Format	20%	Yes	Yes	No	Yes
Identifier Size (# characters)	10%	40	31	6	127
Mnemonic Statement Labels	7%	Yes	Yes	No	Yes
Literal Strings	6%	Yes	Yes	Yes	Yes
Statement Terminator	5%	end-of-line	;	end-of-line	;
Reserved Words	5%	Yes	Yes	Yes	Yes
Nested Comments	5%	end-of-line	Yes	No	Yes
	100%				

TABLE 2 - CONTROL OF DATA

Language Features Listed in order of % importance		Microsoft QuickBASIC	Microsoft C	Microsoft FORTRAN	Turbo Pascal
Local Variables	30%				
to Procedure		Yes	Yes	Yes	Yes
to Block		No	Yes	No	No
Type Checking	20%	Weak	Strong	Weak	Strong
Non-Local Variables	18%	Yes	Yes	Yes	Yes
Language Classification	12%	Static	Stack Based	Static	Stack Based
External Data	10%	Yes	Yes	Yes	No
Data Abstraction	5%				
User Named Types		No	Yes	No	Yes
Type-Checking		No	No	No	Yes
Coercions	4%	Many	Few	Many	Few
Global Variables	1%	Yes	Yes	Yes	Yes
	100%				

TABLE 3 - CONTROL OF EXECUTION

Language Features Listed in order of % importance		Microsoft QuickBASIC	Microsoft C	Microsoft FORTRAN	Turbo Pascal
Procedures	50%				
External or Included		Yes	Yes	Yes	Yes
Internal		Yes	Yes	Yes	Yes
Functions	30%				
External or Included		Yes	Yes	Yes	Yes
Blocks	10%	No	Yes	No	Yes
Local Data		-	Yes	-	No
*Functions					
Internal		Yes	No	Yes	Yes
Recursion	5%	No	Yes	No	Yes
Exception Handling	5%	Yes	No	Yes (minimal)	No
	100%				

* - percentage included with functions

TABLE 4 - PARAMETER PASSING

Language Features Listed in order of % importance		Microsoft QuickBASIC	Microsoft C	Microsoft FORTRAN	Turbo Pascal
Pass by Reference	60%	Yes	No (not directly)	Yes	Yes
Pass by Value	30%	Yes	Yes	No	Yes
Pass by Result	5%	No	No	Yes*	Yes*
Pass by Value-Result	5%	No	No	No	No
	100%				

(* for functions)

TABLE 5 - DATA TYPES AND MANIPULATION

Language Features Listed in order of % importance		Microsoft QuickBASIC	Microsoft C	Microsoft FORTRAN	Turbo Pascal
Numerics	50%				
Real					
Extended		Yes	Yes	Yes	No
Single		Yes	Yes	Yes	Yes
Operations		Yes	Yes	Yes	Yes
Assignment		Yes	Yes	Yes	Yes
Add		Yes	Yes	Yes	Yes
Subtract		Yes	Yes	Yes	Yes
Multiply		Yes	Yes	Yes	Yes
Divide		Yes	Yes	Yes	Yes
Integer		Yes	Yes	Yes	Yes
Functions		Many	None*	Many	Few
Raise to Power		Yes	No*	Yes	Yes
Complex		No	No	Yes	No
Arrays	20%				
Dimensions		63	No Limit	7	No Limit
Subscript Type		Integer	Integer	Integer	Discret
Files	15%				
External		Yes	Yes	Yes	Yes
Sequential		Yes	Yes	Yes	Yes
Direct		Yes	No	Yes	Yes
Internal		Yes	No	Yes	Yes
Miscellaneous	8%				
Boolean Type		No	No	Yes	Yes
Fixed Point		Yes	No	No	No
Record		No	Limited	No	Yes
Pointers		No	Yes	No	Yes
Sets		No	No	No	Yes
Character	7%				
Strings		Yes	No	Yes	No
Arrays		No	Yes	No	Yes
Fixed or Variable length		Variable	Variable (in Fixed array)	Fixed	Fixed
	100%				

(* available in libraries)

TABLE 6 - ADDITIONAL LANGUAGE FACTORS

Language Features Listed in order of importance		Microsoft QuickBASIC	Microsoft C	Microsoft FORTRAN	Turbo Pascal
Libraries	35%	Yes	Yes	Yes	No
Graphics	25%	Yes	No	No	Yes
Multi-Language Usage	10%	Yes	Yes	Yes	No
Overlays	10%	No	No	No	Yes
Chaining	8%	Yes	No	No	No
Database	6%	No	No	No	No
Concurrency	6%	No	No	No	No
	100%				

TABLE 7 - APPLICATION REQUIREMENTS

Application Requirements Listed in order of % importance		Microsoft QuickBASIC	Microsoft C	Microsoft FORTRAN	Turbo Pascal
Functional Areas	30%				
Math, Science		Yes	No	Yes	No
Education		Yes	No	Yes	Yes
Systems		No	Yes	No	No
Reliability	15%	Strong Support	Poor Support	Poor Support	Moderate Support
Portability	14%	Not Suitable	Moderately Suitable	Highly Suitable	Highly Suitable
Memory Limitations	13%				
Code		64K	64K	64K	64K
Data		64K (Each array may take 64K)	640K	640K	64K
Size and Complexity	9%	Small	Moderate	Moderate	Moderate
Expertise	7%	Casual	Moderate	Casual	Moderate
End-user Interaction	6%	Strong Support	Moderate Support	Moderate Support	Moderate Support
Timeframe	6%	Short	Moderate	Moderate	Long
	100%				

TABLE 8 - SUMMARY OF RESULTS

Comparison Factors	Microsoft QuickBASIC	Microsoft C	Microsoft FORTRAN	Turbo Pascal
Syntactic Style				XXX
Control of Data		XXX		
Control of Execution		XXX		
Parameter Passing	XXX			XXX
Data Types and Manipulation			XXX	
Additional Language Factors	XXX			
Application Requirements			XXX	
Performance Test		XXX	XXX	
Practical Considerations			XXX	
Structured Programming		XXX		XXX

REFERENCES

American National Standards Committee X3. 1982. *American National Dictionary for Information Processing Systems*. Computer and Business Equipment Manufacturers Association, Washington, D.C.

American National Standards Institute. 1978. *American National Standard for Minimal BASIC*. American National Standards Institute, New York.

American National Standards Institute and Institute of Electrical and Electronics Engineers. 1983. *American National Standard Pascal Computer Programming Language*. Institute of Electrical and Electronics Engineers, New York.

Barron, D.W. 1977. *An Introduction to the Study of Programming Languages*. Cambridge University Press, England.

Basili, Victor R., and Terry Baker. 1975. *Structured Programming*. Institute of Electrical and Electronics Engineers, New York.

Bates, D. (ed.). 1976. *Infotech State of the Art Report, Structured Programming*. Infotech International Limited, England.

Bathe, K.J. 1982. *Finite Element Procedures in Engineering Analysis*. Prentice-Hall, Englewood Cliffs, New Jersey.

Blankenship, J. 1987. *Structured BASIC Programming with Technical Applications*. Simon and Schuster, Englewood Cliffs, New Jersey.

Boehm, Barry W. 1976. "Software Engineering" in *IEEE Transactions on Computers, C25, No.12*. Institute of Electrical and Electronics Engineers, New York.

Borland International. 1986. *Turbo Pascal*. Scotts Valley, California.

Bredt, Thomas H. 1976. "What the Software Engineeer Should Know about Program Verification", in *Software Engineering Education*. Springer-Verlag, New York.

Brown, Douglas L. 1985. *Pascal to C*. Wadsworth Publishing Company, Belmont, California.

Brown, R.R. 1978. "The Technique and Practice of Structured Design ala Constantine" in *Infotech State of the Art Report, Volume 2, Invited Papers*. Infotech International, England.

Bullen, R.H. 1976. Excerpts from "Engineering of Quality Software Systems: Software Firsts Concepts" in *Structured Programming, Infotech State of the Art Report*. Infotech International, England.

Chapin, N. 1978. "Function Parsing in Structured Design" in *Infotech State of the Art Report, Volume 2, Invited Papers*. Infotech International, England.

Chapra, Steven C, and Raymond P. Canale. 1986. *Introduction to Computing for Engineers*. McGraw Hill, New York, New York.

Coepetz, H. 1979. *Software Reliability*. MacMillan Press Limited, London.

Cook, Robert D. 1981. *Concepts and Applications of Finite Element Analysis*. John Wiley & Sons, New York, New York.

Cugini, John V. 1981. *Specifications and Test Methods for Numeric Accuracy in Programming Language Standards*. National Bureau of Standards, Washington, D.C.

Cugini, John V. 1984. *Selection and Use of General Purpose Programming Languages - Program Examples*. National Bureau of Standards, Washington, D.C. 296

Denning, P.J. 1976. "A Hard Look at Structured Programming" in *Structured Programming, Infotech State of the Art Report*. Infotech International, England.

Dahl, O.J., and E.W. Dijkstra, and C.A.R. Hoare. 1972. *Structured Programming*. Academic Press, New York.

Ghezzi, Carlo, and Mehdi Jazayeri. 1987. *Programming Language Concepts*. John Wiley & Sons, New York.

Gill, S. 1976. Excerpts from "Thoughts on the Sequence of Writing Software" in *Structured Programming, Infotech State of the Art Report*. Infotech International, England.

Gordon, Michael J.C. 1979. *Denotational Description of Programming Languages*. Springer-Verlag, New York.

Griffiths, S.N. 1978. "Design Methodologies - A Comparison" in *Infotech State of the Art Report, Volume 2, Invited Papers*. Infotech International, England.

Higman, Bryan, 1967. *Comparative Study of Programming Languages*. Elsevier Publishing Company, New York.

Hill, I.D., and B.L. Meek. 1980. *Programming Language Standardization*. Ellis Horwood Limited, West Sussex, England.

Holzer, Siegfried M., 1985. *Computer Analysis of Structures*. Elsevier Science Publishing Company, New York.

Huang, J.C. 1976. Excerpts from "An Approach to Program Testing" in *Structured Programming, Infotech State of the Art Report*. Infotech International, England.

Kemeny, John G., and Thomas E. Kurtz. 1985. *Back to BASIC*. Addison-Wesley Publishing Company, Reading, Massachusetts.

Kernighan, Brian W., and Dennis M. Ritchie. 1978. *The C Programming Language*. Prentice Hall, Englewood Cliffs, New Jersey.

Kernighan, Brian W., and P.J. Plauger. 1974. *The Elements of Programming Style*. McGraw Hill, New York.

Ledgard, Henry F., and Michael Marcotty, "A Geneology of Control Structures" in *Communication of the ACM*.

Lee, J.A.N. 1972. *Computer Semantics*. Van Nostrand Reinhold Company, New York.

Linger, R.C., and H.D.Mills, and B.I. Witt. 1979. *Structured Programming: Theory and Practice*. Addison- Wesley Publishing Company, Reading, Massachusetts.

MacLennan, Bruce J.. 1983. *Principles of Programming Languages: Design, Evaluation, and Implementation*. Holt, Reinhart and Winston, New York.

Marcotty, Michael, and Henry F. Ledgard. 1986. *Programming Language Landscape*. Science Research Associates, Chicago.

McClure, C.L. 1976. Excerpts from "Top-down, Bottom-up, and Structured Programming" in *Structured Programming, Infotech State of the Art Report*. Infotech International, England.

McGettrick, Andrew D. 1980. *The Definition of Programming Languages*. Cambridge University Press, England.

McGowan, Clement L., and John R. Kelly. 1975. *Top-Dow Structured Programming Techniques*. Petrocelli/Charter, New York.

Merchant, Michael J. 1981. *FORTRAN 77*, Wadsworth Publishing Company, Belmont, California.

Merchant, Michael J., and John R. Sturgul. 1977. *Applied FORTRAN Programming with Standard FORTRAN, WATFOR, WATFIV, and Structured WATFIV*. Wadsworth Publishing Company, Belmont California.

Microsoft Corporation. 1986. *Microsoft C Compiler Run-Time Library Reference Manual*. Redmond, Washington.

Microsoft Corporation. 1986. *Microsoft C Compiler User's Guide*. Redmond, Washington.

Microsoft Corporation. 1986. *Microsoft CodeView and C Language Reference Manual*. Redmond, Washington.

Microsoft Corporation. 1986. *Microsoft FORTRAN Compiler User's Guide*. Redmond, Washington.

Microsoft Corporation. 1986. *Microsoft FORTRAN Compiler Reference Manual*. Redmond, Washington.

Microsoft Corporation. 1986. *Microsoft QuickBASIC*. Microsoft Corporation, Redmond, Washington.

Organick, Elliot I. 1972. *The MULTICS System*. Massachusetts Institute of Technology, Cambridge, Massachusetts.

Perrott, Ronald H., and Donald C.S. Allison. 1984. *Pascal for FORTRAN Programmers*. Computer Science Press, Rockville, Maryland.

Pratt, Terrence W. 1975. *Programming Languages: Design and Implementation*. Prentice Hall, Englewood Cliffs, New Jersey.

Purdum, J. 1985. *C Self Study Guide*. Que Corporation, Indianapolis, Indiana.

Ramamoorthy, C.V., and R.C. Cheung, and K.H. Kim. 1976. Excerpts of "Reliability and Integrity of Large Computer Programs" in *Structured Programming, Infotech State of the Art Report*. Infotech International, England.

Schneider, Michael G., Steven W. Weingart, and David M. Perlman. 1982. *An Introduction to Programming and Problem Solving in Pascal*. John Wiley & Sons, New York.

Stoy, Joseph E. 1977. *Denotational Semantics: The Scott-Strachy Approach to Programming Language Theory*. Massachusetts Institute of Technology Press, Cambridge, Massachusetts.

Swann, Gloria Harrington. 1978. *Top-Down Structured Design Techniques*. Petrocelli Books, New York.

Tassel, Dennie Van. 1974. *Programming Style, Design, Efficiency, Debugging, and Testing*. Prentice Hall, Englewood Cliffs, New Jersey.

Tennent, R.D. 1981. *Principles of Programming Languages*. Prentice Hall, Englewood Cliffs, New Jersey.

Traister, Robert J. 1985. *Going From BASIC to C*. Prentice Hall, Englewood Cliffs, New Jersey.

Tucker, Allan B. 1977. *Programming Languages*. McGraw Hill, New York.

Verrand, D.Le. 1985. *Evaluating ADA*. North Oxford Academic, Oxford, England.

Wasserman, Anthony I. and Peter Freeman. 1976. *Software Engineering Education*. Springer-Verlag, New York.

Wirth, Niklaus 1973. *Systematic Programming: An Introduction*. Prentice Hall, Englewood Cliffs, New Jersey.

Yourdon, Edward. 1975. *Techniques of Program Structure and Design*. Prentice Hall., Englewood Cliffs, New Jersey.

Yourdon, Edward. 1979. *Structured Walkthroughs*. Prentice Hall, Englewood Cliffs, New Jersey.

APPENDIX A - FORTRAN LISTING

```
C***********************************************************************
C*                              DRIVER                                 *
C***********************************************************************
C
C     PURPOSE: TO OPEN FILES AND INITIALIZE THE VARIABLES
C                NEEDED IN THE INPUT, QINPUT, SKYLINE,
C                AND SOLVE PROCEDURES.
C
C
C-----------------------
C  INPUT REQUIREMENTS
C-----------------------
C LIST DIRECTED IN FILE 'SKY.DAT'
C  1) ENTER NUMBER OF ELEMENTS, NUMBER OF JOINTS, NUBER OF EQUATIONS,
C      AND NUMBER OF LOAD CONDITIONS
C            NE,NJ,NEQ,NLC
C  2) ENTER MEMBER CODE MATRIX ELEMENT BY ELEMENT (i.e., 6 COLUMNS
C      AND NE ROWS)
C          FOR I=1,NE
C            MCODE((I,J),J=1,6)
C          CONTINUE
C  3) ENTER STIFFNESS MATRIX
C           SS(1)     SS(2)     SS(3)
C           SS(4)     SS(5)     SS(6)
C           SS(7)     ...       ...
C            ...      ...       ...
C
C  4) DO FOR EACH LOAD CONDITION
C        ENTER FORCE VECTOR Q
C           Q(1)      Q(2)      Q(3)
C           Q(4)      Q(5)      Q(6)
C           Q(7)      ...       ...
C            ...      ...       ...
C
C     END DO
C
C
C
C
C
C
C-----------------------
C  IDENTIFIERS
C-----------------------
C      A(LIM)       MEMORY BLOCK                             REAL
C      LIM          NUMBER OF MEMORY LOCATIONS ALLOTTED      INTEGER
C      NE           NUMBER OF ELEMENTS                       INTEGER
C      NJ           NUMBER OF JOINTS                         INTEGER
C      NEQ          NUMBER OF EQUATIONS (D.O.F.)             INTEGER
C      LC           LOAD CONDITIONS
C      KEEPNEQ      RETAINS VALUE OF NEQ                     INTEGER
```

```
C       KHT(3*NJ)     COLUMN HEIGHT OF FULL TANGENT        INTEGER
C                       STIFFNESS FOR DEGREE OF
C                       FREEDOM I
C       MAXA(NEQ+1)   STORES ADDRESSES OF DIAGONAL         INTEGER
C                       ELEMENTS OF K
C                       (SYSTEM STIFFNESS MATRIX) IN
C                       ITS COLUMN VECTOR REPRESENTATION
C       MCODE(6,*)    MEMBER CODE MATRIX                   INTEGER
C                       MCODE(I,J) IS THE DEGREE OF
C                       FREEDOM NUMBER IN THE Ith
C                       GLOBAL DIRECTION OF MEMBER J
C       LSS           LENGTH OF SS                         INTEGER
C
C
C
C
C-----------------------
C CONVENTION FOR POINTERS
C-----------------------
C NNAME   LOCATION OF FIRST ELEMENT OF ARRAY
C           NAME IN A(LIM)
C         WHILE SOME ARE NOT USED IN THIS IMPLEMENTATION,
C           THEY ARE DEFINED FOR FUTURE EXPANSION
C
C       NP            SET = 1, FIRST LOCATION OF           INTEGER
C                       P IN A(LIM).
C       NAREA         SET = NP+3*NJ, FIRST LOCATION OF     INTEGER
C                       AREA IN A(LIM).
C       NZI           SET = NAREA+NE, FIRST LOCATION OF    INTEGER
C                       ZI IN A(LIM)
C       NEMOD         SET = NZI+NE, FIRST LOCATION OF      INTEGER
C                       EMOD IN A(LIM)
C       NCTE          SET = NEMOD+NE, FIRST LOCATION OF    INTEGER
C                       CTE IN A(LIM)
C       NELENG        SET = NCTE+NE, FIRST LOCATION OF     INTEGER
C                       ELENG IN A(LIM)
C       NC1           SET = NELENG+NE, FIRST LOCATION OF   INTEGER
C                       C1 IN A(LIM)
C       NC2           SET= NC1+NE, FIRST LOCATION OF       INTEGER
C                       C2 IN A(LIM)
C       NMCODE        SET= NCE+NE, FIRST LOCATION OF       INTEGER
C                       MCODE IN A(LIM)
C       NJCODE        SET= NMCODE+6*NE, FIRST LOCATION OF  INTEGER
C                       JCODE IN A(LIM)
C       NMINC         SET= NJCODE+3*NJ, FIRST LOCATION OF  INTEGER
C                       MINC IN A(LIM)
C       NMAXA         SET= NMINC+2*NE, FIRST LOCATION OF   INTEGER
C                       MAXA IN A(LIM)
C       NKHT          SET= NMAXA+NEQ+1, FIRST LOCATION OF  INTEGER
C                       KHT IN A(LIM)
C       NF            SET= NMAXA+NEQ+1, FIRST LOCATION OF  INTEGER
C                       F IN A(LIM)
C       NSS           SET= NF+6*NE, FIRST LOCATION OF      INTEGER
C                       SS IN A(LIM)
C       NQ            SET= NSS+LSS, FIRST LOCATION OF      INTEGER
C                       Q IN A(LIM)
C       NNA           SET= NQ+NEQ, FIRST LOCATION OF       INTEGER
```

```
C                         NA IN A(LIM)
C      MAX              SET= NNA+NE-1, LAST LOCATION IN      INTEGER
C                         A(LIM)
C
C
      PARAMETER (LIM=10000)
      DIMENSION A(LIM)
C
      OPEN (5,FILE='SKY.DAT')
      OPEN (6,FILE='SKY.OUT',STATUS='NEW')
C
C READ AND ECHO NE,NJ,NEQ,LC
C
      READ (5,*) NE,NJ,NEQ,NLC
      WRITE (6,'(4(2X,A,I3/))') 'NE=',NE,'NJ=',NJ,'NEQ=',NEQ,'LC=',NLC
      KEEPNEQ=NEQ
C
C  INITIALIZE POINTERS OF DATA INTO A(LIM)
C
      NP=1
      NAREA=NP+3*NJ
      NZI=NAREA+NE
      NEMOD=NZI+NE
      NCTE=NEMOD+NE
      NELENG=NCTE+NE
      NC1=NELENG+NE
      NC2=NC1+NE
      NMCODE=NC2+NE
      NJCODE=NMCODE+6*NE
      NMINC=NJCODE+3*NJ
      NMAXA=NMINC+2*NE
C
C TEMPORARILY SET THESE, SINCE CURRENTLY UNKNOWN
C
      NEQ=3*NJ
      NKHT=NMAXA+(NEQ+1)
      MAX=NKHT+NEQ-1
C
C IF MEMORY NOT EXCEEDED CALL INPUT AND SKYLIN ELSE PRINT MESSAGE
C

      IF (MAX.LE.LIM) THEN
        CALL INPUT(A(NMCODE),NE)
        NEQ=KEEPNEQ
        CALL SKYLIN(A(NKHT),A(NMAXA),A(NMCODE),NE,NEQ,LSS)
      ELSE
        WRITE (6,'(T10,A/)') 'ERROR MESSAGE: INCREASE MEMORY'
      ENDIF
C
C RESET TEMPORARY POINTERS SINCE PARAMETERS NOW KNOWN
C
      NF=NMAXA+(NEQ+1)
      NSS=NF+6*NE
      NQ=NSS+LSS
      NNA=NQ+NEQ
      MAX=NNA+NE-1
C
```

```
C IF MEMORY NOT EXCEEDED CALL QINPUT AND SOLVE ELSE PRINT MESSAGE
C
      IF (MAX.LE.LIM) THEN
        CALL SSINPUT(A(NF),LSS)
        DO 10 LC=1,NLC
          WRITE (6,'(//T10,A,I3//)') 'LOAD CONDITION ',LC
          CALL QINPUT(A(NQ),NEQ)
          CALL SOLVE(A(NF),A(NQ),A(NMAXA),NEQ,LC)
   10   CONTINUE
      ELSE
        WRITE (6,'(T10,A/)') 'ERROR MESSAGE: INCREASE MEMORY'
      ENDIF
C
      STOP
      END
C**********************************************************************
C*                                INPUT                               *
C**********************************************************************
C
C     PURPOSE: INPUT AND ECHO MCODE(6,*)
C
C     INPUT : NE
C     OUTPUT: MCODE(6,*)
C
C
C----------------------
C    IDENTIFIERS
C----------------------
C
C      NE            NUMBER OF ELEMENTS                     INTEGER
C      MCODE(6,*)    MEMBER CODE MATRIX                     INTEGER
C                      MCODE(I,J) IS THE DEGREE OF
C                      FREEDOM NUMBER IN THE Ith
C                      GLOBAL DIRECTION OF MEMBER J
C
        SUBROUTINE INPUT(MCODE,NE)
        DIMENSION MCODE(6,*)
        WRITE (6,'(//A/)') 'MCODE TRANSPOSED'
        DO 10 I=1,NE
          READ (5,*) (MCODE(J,I),J=1,6)
          WRITE (6,'(6(1X,I4,1X)')) (MCODE(J,I),J=1,6)
   10   CONTINUE
        RETURN
        END
C**********************************************************************
C*                               SSINPUT                              *
C**********************************************************************
C
C     PURPOSE: INPUT AND ECHO SS
C
C     INPUT :LSS
C     OUTPUT:SS(*)
C
C----------------------
C  IDENTIFIERS
C----------------------
C
```

```
C      LSS           NUMBER OF ELEMENTS IN SS                  INTEGER
C      SS(LSS)       SYSTEM STIFNESS MATRIX                    REAL
C
        SUBROUTINE SSINPUT(SS,LSS)
        DIMENSION SS(*)
        READ (5,*) (SS(I),I=1,LSS)
          WRITE (6,10)  (I,SS(I),I=1,LSS)
   10     FORMAT (//T10,'SS MATRIX'/3(2X,'SS(',I5,')= ',F14.7))
        RETURN
        END
C***********************************************************************
C*                                QINPUT                               *
C***********************************************************************
C
C     PURPOSE: INPUT AND ECHO Q
C
C     INPUT :NEQ
C     OUTPUT:Q(*)
C
C----------------------
C  IDENTIFIERS
C----------------------
C
C      NEQ           NUMBER OF EQUATIONS                       INTEGER
C      Q(NEQ)        DISPLACEMENTS                             REAL
C
        SUBROUTINE QINPUT(Q,NEQ)
        DIMENSION Q(*)
         READ (5,*) (Q(I),I=1,NEQ)
         WRITE (6,20) (I,Q(I),I=1,NEQ)
   20   FORMAT (//T10,'FORCES'/3(2X,'Q(',I5,')= ',F14.7))
        RETURN
        END
C***********************************************************************
C*                                 SKYLIN                              *
C***********************************************************************
C
C     PURPOSE: SKYLINE DETERMINES KHT USING MCODE, AND
C                   DETERMINES MAXA FROM KHT.
C
C     INPUT : MCODE(6,*),NE,NEQ
C     OUTPUT: KHT(*),MAXA(*),LSS
C
C----------------------
C  IDENTIFIERS
C----------------------
C
C      NE            NUMBER OF ELEMENTS                        INTEGER
C      NEQ           NUMBER OF EQUATIONS (D.O.F.)              INTEGER
C      KHT(I)        COLUMN HEIGHT OF FULL TANGENT             INTEGER
C                      STIFFNESS FOR DEGREE OF
C                      FREEDOM I
C      MAXA(I)       STORES ADDRESSES OF DIAGONAL              INTEGER
C                      ELEMENTS OF K
C                      (SYSTEM STIFFNESS MATRIX) IN
C                      ITS COLUMN VECTOR REPRESENTATION
C      MCODE(6,*)    MEMBER CODE MATRIX                        INTEGER
```

```
C                        MCODE(I,J) IS THE DEGREE OF
C                        FREEDOM NUMBER IN THE Ith
C                        GLOBAL DIRECTION OF MEMBER J
C       LSS            LAST SS, NUMBER OF ELEMENTS IN        INTEGER
C                        ARRAY SS.
C
C
      SUBROUTINE SKYLIN(KHT,MAXA,MCODE,NE,NEQ,LSS)
      DIMENSION KHT(*),MAXA(*),MCODE(6,*)
      DO 10 I=1,NEQ
         KHT(I)=0
   10 CONTINUE
C
C     GENERATE AND PRINT KHT
C
      DO 30 I=1,NE
         J=1
   15    IF(MCODE(J,I) .EQ. 0 .AND. J .LT. 6) THEN
            J=J+1
            GO TO 15
         ENDIF
         MIN=MCODE(J,I)
         J=J+1
         DO 20 L=J,6
            K=MCODE(L,I)
            IF(K.NE.0) THEN
               KHT(K)=MAX0(KHT(K),(K-MIN))
            ENDIF
   20    CONTINUE
   30 CONTINUE
      WRITE(6,100)
  100 FORMAT(//11X,'I',10X,'KHT(I)',10X,'MAXA(I)')
C
C     GENERATE AND PRINT MAXA
C
      MAXA(1)=1
      DO 40 I=1,NEQ
         WRITE(6,200) I,KHT(I),MAXA(I)
  200    FORMAT(7X,I5,9X,I5,11X,I5)
         MAXA(I+1)=MAXA(I)+KHT(I)+1
   40 CONTINUE
      LSS=MAXA(NEQ+1)-1
      I=NEQ+1
      WRITE(6,300) I,MAXA(I),LSS
  300 FORMAT(7X,I5,25X,I5//7X,'LSS = ',I5)
      RETURN
      END
C***************************************************************************
C*                                  SOLVE                                  *
C***************************************************************************
C
C     PURPOSE: SOLVE DETERMINES THE SOLUTION TO THE SYSTEM EQUATIONS
C                 BY COMPACT GAUSSIAN ELIMINATION
C                 (HOLZER, PP. 290, 296, 307) BASED ON THE SUBROUTINE
C                 COLSOL (BATHE P. 721) AND THE MODIFICATION BY MICHAEL
C                 BUTLER (MS 1984): FOR THE FIRST LOAD CONDITION,
C                 LC = 1, CALL FACTOR,REDUCE,AND BACSUB; FOR
```

```
C                 SUBSEQUENT LOAD CONDITIONS, LC > 1, CALL REDUCE AND
C                 BACSUB.
C
C      INPUT : NEQ,LC,MAXA(*),SS(*),Q(*)
C      OUTPUT: SS(*),Q(*)
C
C-----------------------
C  IDENTIFIERS
C-----------------------
C
C       LC           LOAD CONDITION
C       NEQ          NUMBER OF EQUATIONS (D.O.F.)          INTEGER
C       MAXA(I)      STORES ADDRESSES OF DIAGONAL          INTEGER
C                      ELEMENTS OF K
C                      (SYSTEM STIFFNESS MATRIX) IN
C                      ITS COLUMN VECTOR REPRESENTATION
C       SS(*)        VECTOR REPRESENTATION OF              REAL
C                      SYSTEM STIFFNESS MATRIX
C       Q(NEQ)       APPLIED FORCE DISTRIBUTION            REAL
C                        (UNFACTORED)
C
C
C
C
C    LC = 1, SYSTEM NOT INITIALIZED BY FACTOR,...CALL FACTOR
C    LC >= 1, SYSTEM INITIALIZED, CALL REDUCE AND BACSUB
C
C
      SUBROUTINE SOLVE(SS,Q,MAXA,NEQ,LC)
      DIMENSION SS(*),Q(*),MAXA(*)
C
C WRITE 'BEGIN' TO SCREEN FOR TIMING AND OPEN FILES
C
      WRITE (*,'(//T5,A,I3//)') 'LOAD CONDITION ',LC
      WRITE (*,*) 'BEGIN'
      IF (LC.EQ.1) THEN
          CALL FACTOR(SS,MAXA,NEQ)
      END IF
      CALL REDUCE(SS,Q,MAXA,NEQ)
      CALL BACSUB(SS,Q,MAXA,NEQ)
C
C WRITE 'END' TO SCREEN FOR TIMING
C
      WRITE (*,*) 'END'
C
C PRINT RESULTANT DISPLACEMENTS,Q
C
        WRITE (6,10) (I,Q(I),I=1,NEQ)
  10    FORMAT(/T10,'RESULTANT DISPLACEMENTS'/3(2X,'Q(',I4,')= ',F14.7))
      RETURN
      END
C**************************************************************************
C*                                  FACTOR                                *
C**************************************************************************
C
C      PURPOSE:  FACTOR PERFORMS THE LDU FACTORIZATION OF
C                 THE STIFFNESS MATRIX.
```

```
C
C     INPUT : NEQ,MAXA(*),SS(*)
C     OUTPUT: SS(*)
C
C------------------------
C  IDENTIFIERS
C------------------------
C
C      NEQ          NUMBER OF EQUATIONS (D.O.F.)          INTEGER
C      MAXA(I)      STORES ADDRESSES OF DIAGONAL          INTEGER
C                     ELEMENTS OF K
C                     (SYSTEM STIFFNESS MATRIX) IN
C                     ITS COLUMN VECTOR REPRESENTATION
C      SS(*)        VECTOR REPRESENTATION OF              REAL
C                     SYSTEM STIFFNESS MATRIX
C
      SUBROUTINE FACTOR(SS,MAXA,NEQ)
      DIMENSION SS(*),MAXA(*)
C
      DO 80 N=1,NEQ
        KN=MAXA(N)
        KL=KN+1
        KU=MAXA(N+1)-1
        KH=KU-KL
        IF (KH.GT.0) THEN
          K=N-KH
          IC=0
          KLT=KU
          DO 40 J=1,KH
              IC=IC+1
              KLT=KLT-1
              KI=MAXA(K)
              ND=MAXA(K+1)-KI-1
              IF (ND.GT.0) THEN
                  IF ((IC-ND).GT.0) THEN
                     KK=ND
                  ELSE
                     KK=IC
                  ENDIF
                  C=0.00
                  DO 30 L=1,KK
                    C=C+SS(KI+L)*SS(KLT+L)
   30             CONTINUE
                  SS(KLT)=SS(KLT)-C
              ENDIF
              K=K+1
   40     CONTINUE
        ENDIF
        IF (KH.GE.0) THEN
   50     K=N
          B=0.00
          DO 60 KK=KL,KU
            K=K-1
            KI=MAXA(K)
            C=SS(KK)/SS(KI)
            B=B+C*SS(KK)
            SS(KK)=C
   60     CONTINUE
          SS(KN)=SS(KN)-B
        ENDIF
C
C     STOP EXECUTION IF A ZERO PIVOT IS DETECTED
C
   70   IF(SS(KN).EQ.0.00) THEN
          WRITE (6,75) N,SS(KN)
   75     FORMAT('-STIFFNESS MATRIX IS NOT POSITIVE DEFINITE'/'0PIVOT IS
     $    ZERO FOR D.O.F. ',I4/'0PIVOT = ',E15.8)
          STOP
        END IF
C
   80 CONTINUE
      RETURN
      END
C
C***********************************************************************
C*                                REDUCE                               *
C***********************************************************************
C
C     PURPOSE: REDUCE REDUCES THE RIGHT-SIDE LOAD VECTOR.
C
C     INPUT : MAXA(*),SS(*),Q(*),NEQ
C     OUTPUT: Q(*)
C
C----------------------
C  IDENTIFIERS
C----------------------
C
C      NEQ           NUMBER OF EQUATIONS (D.O.F.)          INTEGER
C      MAXA(I)       STORES ADDRESSES OF DIAGONAL          INTEGER
C                      ELEMENTS OF K
C                      (SYSTEM STIFFNESS MATRIX) IN
C                      ITS COLUMN VECTOR REPRESENTATION
C      SS(*)         VECTOR REPRESENTATION OF            . REAL
C                      SYSTEM STIFFNESS MATRIX
C      Q(NEQ)        APPLIED FORCE DISTRIBUTION            REAL
C                        (UNFACTORED)
C
C
C
      SUBROUTINE REDUCE(SS,Q,MAXA,NEQ)
      DIMENSION SS(*),Q(*),MAXA(*)
C
      DO 20 N=1,NEQ
        KL=MAXA(N)+1
        KU=MAXA(N+1)-1
        KH=KU-KL
        IF (KH.GE.0) THEN
          K=N
          C=0.00
          DO 10 KK=KL,KU
            K=K-1
            C=C+SS(KK)*Q(K)
   10     CONTINUE
          Q(N)=Q(N)-C
          END IF
   20  CONTINUE
       RETURN
       END
C
C*************************************************************************
C*                                BACSUB                                 *
C*************************************************************************
C
C      PURPOSE: BACSUB PERFORMS BACK-SUBSTITUTION TO OBTAIN
C                  THE SOLUTION.
C
C      INPUT : NEQ,MAXA(*),SS(*),Q(*)
C      OUTPUT: Q(*)
C
C-----------------------
C  IDENTIFIERS
C-----------------------
C
C        NEQ           NUMBER OF EQUATIONS (D.O.F.)           INTEGER
C        MAXA(I)       STORES ADDRESSES OF DIAGONAL           INTEGER
C                        ELEMENTS OF K
C                        (SYSTEM STIFFNESS MATRIX) IN
C                        ITS COLUMN VECTOR REPRESENTATION
C        SS(*)         VECTOR REPRESENTATION OF               REAL
C                        SYSTEM STIFFNESS MATRIX
C        Q(NEQ)        APPLIED FORCE DISTRIBUTION             REAL
C                           (UNFACTORED)
C
       SUBROUTINE BACSUB(SS,Q,MAXA,NEQ)
       DIMENSION SS(*),Q(*),MAXA(*)
C
       DO 10 N=1,NEQ
         K=MAXA(N)
         Q(N)=Q(N)/SS(K)
   10  CONTINUE
       IF (NEQ.GT.1) THEN
         N=NEQ
         DO 30 L=2,NEQ
           KL=MAXA(N)+1
           KU=MAXA(N+1)-1
           KH=KU-KL
           IF (KH.GE.0) THEN
             K=N
             DO 20 KK=KL,KU
               K=K-1
               Q(K)=Q(K)-SS(KK)*Q(N)
   20        CONTINUE
           END IF
           N=N-1
   30    CONTINUE
       ENDIF
       RETURN
       END
```

APPENDIX B - QUICKBASIC LISTING

```
'***********************************************************************
'*                              DRIVER                                 *
'***********************************************************************
'
'     PURPOSE: TO OPEN FILES AND INITIALIZE THE VARIABLES
'              NEEDED IN THE INPUT, QINPUT, SKYLINE,
'              AND SOLVE PROCEDURES.
'
'
'----------------------
'  INPUT REQUIREMENTS
'----------------------
' LIST DIRECTED IN FILE 'SKY.DAT'
'  1) ENTER NUMBER OF ELEMENTS, NUMBER OF JOINTS, NUBER OF EQUATIONS,
'      AND NUMBER OF LOAD CONDITIONS
'            NE,NJ,NEQ,NLC
'  2) ENTER MEMBER CODE MATRIX ELEMENT BY ELEMENT (i.e., 6 COLUMNS
'      AND NE ROWS)
'          FOR I=1,NE
'            MCODE((I,J),J=1,6)
'          CONTINUE
'  3) ENTER STIFFNESS MATRIX
'           SS(1)      SS(2)      SS(3)
'           SS(4)      SS(5)      SS(6)
'           SS(7)       ...        ...
'            ...        ...        ...
'
'  4) DO FOR EACH LOAD CONDITION
'        ENTER FORCE VECTOR Q
'           Q(1)      Q(2)      Q(3)
'           Q(4)      Q(5)      Q(6)
'           Q(7)       ...       ...
'            ...       ...       ...
'
'     END DO
'
'
'
'
'
'
'----------------------
'  IDENTIFIERS
'----------------------
'      NE           NUMBER OF ELEMENTS                  INTEGER
'      NJ           NUMBER OF JOINTS                    INTEGER
'      NEQ          NUMBER OF EQUATIONS (D.O.F.)        INTEGER
'      LC           LOAD CONDITIONS
'      KEEPNEQ      RETAINS VALUE OF NEQ                INTEGER
'      KHT(3*NJ)    COLUMN HEIGHT OF FULL TANGENT       INTEGER
'                       STIFFNESS FOR DEGREE OF
'                       FREEDOM I
'     MAXA(NEQ+1)  STORES ADDRESSES OF DIAGONAL         INTEGER
'                       ELEMENTS OF K
'                       (SYSTEM STIFFNESS MATRIX) IN
'                       ITS COLUMN VECTOR REPRESENTATION
'     MCODE(6,*)   MEMBER CODE MATRIX                   INTEGER
'                       MCODE(I,J) IS THE DEGREE OF
'                       FREEDOM NUMBER IN THE Ith
'                       GLOBAL DIRECTION OF MEMBER J
'     LSS          LENGTH OF SS                         INTEGER
'
'
'
' OPEN FILES FOR INPUT AND OUTPUT
'
    OPEN "I",#1,"SKY.DAT"
    OPEN "O",#2,"SKY.OUT
'
' READ AND ECHO NE,NJ,NEQ,LC
'
    INPUT #1, NE,NJ,NEQ,NLC
    PRINT #2, USING "NE=  ###";NE
    PRINT #2, USING "NJ=  ###";NJ
    PRINT #2, USING "NEQ= ###";NEQ
    PRINT #2, USING "LC=  ###";NLC
    KEEPNEQ=NEQ

'
' NOTE:  QUICKBASIC DOES NOT ALLOW THE PASSING OF ARRAYS IN THE SAME
'         FASHION AS FORTRAN.  THUS, EACH ARRAY MUST BE DIMENSIONED
'         IN THE MAIN PROGRAM AND PASSED TO THE SUBROUTINES.
'
'
    DIM MCODE(6,NE),KHT(NEQ),MAXA(NEQ+1)
'
' CALL INPUT AND SKYLIN
'
    CALL IN(MCODE(),NE)
    NEQ=KEEPNEQ
    CALL SKYLIN(KHT(),MAXA(),MCODE(),NE,NEQ,LSS)
'
' DIMENSION SS AND Q SINCE LENGTH NOW KNOWN
'
    DIM SS(LSS),Q(NEQ)
'
' CALL QINPUT AND SOLVE
'
    CALL SSINPUT(SS(),LSS)
    FOR LC=1 TO NLC
      PRINT #2,:PRINT #2,
      PRINT #2, "  LOAD CONDITION " LC
      PRINT "  LOAD CONDITION ",LC
      CALL QINPUT(Q(),NEQ)
      CALL SOLVE(SS(),Q(),MAXA(),NEQ,LC)
    NEXT LC
    END
```

```
'***************************************************************************
'*                               INPUT                                     *
'***************************************************************************
'
'     PURPOSE: INPUT AND ECHO MCODE(6,*)
'
'     INPUT : NE
'     OUTPUT: MCODE(6,*)
'
'
'-----------------------
'    IDENTIFIERS
'-----------------------
'
'      NE            NUMBER OF ELEMENTS                      INTEGER
'      MCODE(6,*)    MEMBER CODE MATRIX                      INTEGER
'                      MCODE(I,J) IS THE DEGREE OF
'                      FREEDOM NUMBER IN THE Ith
'                      GLOBAL DIRECTION OF MEMBER J
'
       SUB IN(MCODE(2),NE) STATIC
       PRINT #2,:PRINT #2,
       PRINT #2,"MCODE TRANSPOSED"
       FOR I=1 TO NE
         INPUT #1,MCODE(1,I),MCODE(2,I),MCODE(3,I),MCODE(4,I),_
                                        MCODE(5,I),MCODE(6,I)
         PRINT #2, USING "### ### ### ### ### ###";MCODE(1,I),_
                                        MCODE(2,I),MCODE(3,I),_
                                        MCODE(4,I),MCODE(5,I),_
                                        MCODE(6,I)
       NEXT I
       END SUB
'***************************************************************************
'*                              SSINPUT                                    *
'***************************************************************************
'
'     PURPOSE: INPUT AND ECHO SS
'
'     INPUT :LSS
'     OUTPUT:SS(*)
'
'-----------------------
'  IDENTIFIERS
'-----------------------
'
'      LSS           NUMBER OF ELEMENTS IN SS                INTEGER
'      SS(LSS)       SYSTEM STIFNESS MATRIX                  REAL
'
       SUB SSINPUT(SS(1),LSS) STATIC
       SS1$= "SS(####)= ######.######  "
       SS2$=SS1$+SS1$
       SS3$=SS1$+SS1$+SS1$
       PRINT #2,:PRINT #2,
       PRINT #2,"SS MATRIX"
       NUM=INT(LSS/3)
       FOR I=1 TO NUM
          INPUT #1,SS(I*3-2),SS(I*3-1),SS(I*3)
```

```
           PRINT #2, USING SS1$;I*3-2,SS(I*3-2),I*3-1,SS(I*3-1),I*3,SS(I*3)
        NEXT I
        NUM=LSS-(NUM*3)
        IF (NUM <> 0) THEN
          IF (NUM = 1) THEN
             INPUT #1,SS(LSS-NUM+1)
             PRINT #2, USING SS1$;LSS-NUM+1,SS(LSS-NUM+1)
          ELSEIF (NUM = 2) THEN
             INPUT #1,SS(LSS-NUM+1),SS(LSS-NUM+2)
             PRINT #2, USING SS2$;LSS-NUM+1,SS(LSS-NUM+1),_
                                  LSS-NUM+2,SS(LSS-NUM+2)
          END IF
        END IF
        END SUB
'**********************************************************************
'*                               QINPUT                               *
'**********************************************************************
'
'     PURPOSE: INPUT AND ECHO Q
'
'     INPUT :NEQ
'     OUTPUT:Q(*)
'
'-----------------------
'  IDENTIFIERS
'-----------------------
'
'      NEQ          NUMBER OF EQUATIONS                       INTEGER
'      Q(NEQ)       DISPLACEMENTS                             REAL
'
        SUB QINPUT(Q(1),NEQ) STATIC
        Q1$= "Q(###)= ######.######  "
        Q2$=Q1$+Q1$
        Q3$=Q1$+Q1$+Q1$
        PRINT #2,:PRINT#2,
        PRINT #2,"FORCES"
        NUM=INT(NEQ/3)
        FOR I=1 TO NUM
           INPUT #1,Q(I*3-2),Q(I*3-1),Q(I*3)
           PRINT #2, USING Q3$;I*3-2,Q(I*3-2),I*3-1,Q(I*3-1),I*3,Q(I*3)
        NEXT I
        NUM=NEQ-(NUM*3)
        IF (NUM <> 0) THEN
          IF (NUM = 1) THEN
             INPUT #1,Q(NEQ-NUM+1)
             PRINT #2, USING Q1$;NEQ-NUM+1,Q(NEQ-NUM+1)
          ELSEIF (NUM = 2) THEN
             INPUT #1,Q(NEQ-NUM+1),Q(NEQ-NUM+2)
             PRINT #2, USING Q2$;NEQ-NUM+1,Q(NEQ-NUM+1),_
                                 NEQ-NUM+2,Q(NEQ-NUM+2)
          END IF
        END IF
        END SUB
'**********************************************************************
'*                               SKYLIN                               *
'**********************************************************************
'
```

```
'     PURPOSE: SKYLINE DETERMINES KHT USING MCODE, AND
'                DETERMINES MAXA FROM KHT.
'
'     INPUT : MCODE(6,*),NE,NEQ
'     OUTPUT: KHT(*),MAXA(*),LSS
'
'-----------------------
'  IDENTIFIERS
'-----------------------
'
'      NE            NUMBER OF ELEMENTS                      INTEGER
'      NEQ           NUMBER OF EQUATIONS (D.O.F.)            INTEGER
'      KHT(I)        COLUMN HEIGHT OF FULL TANGENT           INTEGER
'                      STIFFNESS FOR DEGREE OF
'                      FREEDOM I
'      MAXA(I)       STORES ADDRESSES OF DIAGONAL            INTEGER
'                      ELEMENTS OF K
'                      (SYSTEM STIFFNESS MATRIX) IN
'                      ITS COLUMN VECTOR REPRESENTATION
'      MCODE(6,*)    MEMBER CODE MATRIX                      INTEGER
'                      MCODE(I,J) IS THE DEGREE OF
'                      FREEDOM NUMBER IN THE Ith
'                      GLOBAL DIRECTION OF MEMBER J
'      LSS           LAST SS, NUMBER OF ELEMENTS IN          INTEGER
'                      ARRAY SS.
'
'
      SUB SKYLIN(KHT(1),MAXA(1),MCODE(2),NE,NEQ,LSS) STATIC
      FOR I=1 TO NEQ
         KHT(I)=0
      NEXT I
'
'     GENERATE AND PRINT KHT
'
      FOR I=1 TO NE
         J=1
         WHILE ( (MCODE(J,I) = 0) AND (J < 6) )
            J=J+1
         WEND
         MIN=MCODE(J,I)
         J=J+1
         FOR L=J TO 6
            K=MCODE(L,I)
            IF (K <> 0) THEN
'  SET KHT(K)=MAX0(KHT(K),(K-MIN))
               IF ( KHT(K) < (K-MIN) ) THEN
                  KHT(K)=K-MIN
               END IF
            END IF
         NEXT L
      NEXT I
      PRINT #2,:PRINT #2,
      PRINT #2, "          I          KHT(I)           MAXA(I)"
'
'     GENERATE AND PRINT MAXA
'
      MAXA(1)=1
```

```
    FOR I=1 TO NEQ
       PRINT #2, USING "       #####         #####          #####";_
                                                     I,KHT(I),MAXA(I)
       MAXA(I+1)=MAXA(I)+KHT(I)+1
    NEXT I
    LSS=MAXA(NEQ+1)-1
    I=NEQ+1
    PRINT #2, USING "       #####                            #####";_
                                                        I,MAXA(I)
    PRINT #2,:PRINT #2,
    PRINT #2, USING "       LSS =                            #####";LSS
    END SUB
'******************************************************************************
'*                                  SOLVE                                     *
'******************************************************************************
'
'   PURPOSE: SOLVE DETERMINES THE SOLUTION TO THE SYSTEM EQUATIONS
'            BY COMPACT GAUSSIAN ELIMINATION
'            (HOLZER, PP. 290, 296, 307) BASED ON THE SUBROUTINE
'            COLSOL (BATHE P. 721) AND THE MODIFICATION BY MICHAEL
'            BUTLER (MS 1984): FOR THE FIRST LOAD CONDITION,
'            LC = 1, CALL FACTOR,REDUCE,AND BACSUB; FOR
'            SUBSEQUENT LOAD CONDITIONS, LC > 1, CALL REDUCE AND
'            BACSUB.
'
'   INPUT : NEQ,LC,MAXA(*),SS(*),Q(*)
'   OUTPUT: SS(*),Q(*)
'
'----------------------
' IDENTIFIERS
'----------------------
'
'    LC          LOAD CONDITION
'    NEQ         NUMBER OF EQUATIONS (D.O.F.)          INTEGER
'    MAXA(I)     STORES ADDRESSES OF DIAGONAL          INTEGER
'                  ELEMENTS OF K
'                  (SYSTEM STIFFNESS MATRIX) IN
'                  ITS COLUMN VECTOR REPRESENTATION
'    SS(*)       VECTOR REPRESENTATION OF              REAL
'                  SYSTEM STIFFNESS MATRIX
'    Q(NEQ)      APPLIED FORCE DISTRIBUTION            REAL
'                    (UNFACTORED)
'
'
'
'
'  LC = 1, SYSTEM NOT INITIALIZED BY FACTOR,...CALL FACTOR
'  LC >= 1, SYSTEM INITIALIZED, CALL REDUCE AND BACSUB
'
'
    SUB SOLVE(SS(1),Q(1),MAXA(1),NEQ,LC) STATIC
'
    Q1$= "Q(###)= ######.#######  "
    Q2$=Q1$+Q1$
    Q3$=Q1$+Q1$+Q1$
    PRINT "    START " TIME$
    PRINT #2,
      PRINT #2, "    START " TIME$
      START!=TIMER
      IF (LC = 1) THEN
          CALL FACTOR(SS(),MAXA(),NEQ)
      END IF
      CALL REDUCE(SS(),Q(),MAXA(),NEQ)
      CALL BACSUB(SS(),Q(),MAXA(),NEQ)
'
' PRINT RESULTANT FORCES,Q
'
        FINISH!=TIMER
        PRINT "    FINISH " TIME$
        PRINT #2, "    FINISH " TIME$
        PRINT "    Time elapsed in Solution Process:" FINISH!-START!_
                                                   "Seconds"
        PRINT #2,"    Time elapsed in Solution Process:" FINISH!-START!_
                                                      "Seconds"
        PRINT #2,:PRINT#2,
        PRINT #2,"DISPLACEMENTS"
        NUM=INT(NEQ/3)
        FOR I=1 TO NUM
           PRINT #2, USING Q3$;I*3-2,Q(I*3-2),I*3-1,Q(I*3-1),I*3,Q(I*3)
        NEXT I
        NUM=NEQ-(NUM*3)
        IF (NUM <> 0) THEN
          IF (NUM = 1) THEN
             PRINT #2, USING Q1$;NEQ-NUM+1,Q(NEQ-NUM+1)
          ELSEIF (NUM = 2) THEN
             PRINT #2, USING Q2$;NEQ-NUM+1,Q(NEQ-NUM+1),_
                                 NEQ-NUM+2,Q(NEQ-NUM+2)
          END IF
        END IF
      END SUB
'***********************************************************************
'*                               FACTOR                               *
'***********************************************************************
'
'     PURPOSE:  FACTOR PERFORMS THE LDU FACTORIZATION OF
'                THE STIFFNESS MATRIX.
'
'     INPUT : NEQ,MAXA(*),SS(*)
'     OUTPUT: SS(*)
'
'-----------------------
'  IDENTIFIERS
'-----------------------
'
'      NEQ          NUMBER OF EQUATIONS (D.O.F.)          INTEGER
'      MAXA(I)      STORES ADDRESSES OF DIAGONAL          INTEGER
'                     ELEMENTS OF K
'                     (SYSTEM STIFFNESS MATRIX) IN
'                     ITS COLUMN VECTOR REPRESENTATION
'      SS(*)        VECTOR REPRESENTATION OF              REAL
'                     SYSTEM STIFFNESS MATRIX
'
      SUB FACTOR(SS(1),MAXA(1),NEQ) STATIC
'
```

```
      FOR N=1 TO NEQ
        KN=MAXA(N)
        KL=KN+1
        KU=MAXA(N+1)-1
        KH=KU-KL
       IF (KH >= 0) THEN
        IF (KH > 0) THEN
          K=N-KH
          IC=0
          KLT=KU
          FOR J=1 TO KH
              IC=IC+1
              KLT=KLT-1
              KI=MAXA(K)
              ND=MAXA(K+1)-KI-1
              IF (ND > 0) THEN
                  IF ((IC-ND) > 0) THEN
                     KK=ND
                  ELSE
                     KK=IC
                  END IF
                  C=0.00
                  FOR L=1 TO KK
                    C=C+SS(KI+L)*SS(KLT+L)
                  NEXT L
                  SS(KLT)=SS(KLT)-C
              END IF
              K=K+1
          NEXT J
       END IF

          K=N
          B=0.00
          FOR KK=KL TO KU
            K=K-1
            KI=MAXA(K)
            C=SS(KK)/SS(KI)
            B=B+C*SS(KK)
            SS(KK)=C
          NEXT KK
          SS(KN)=SS(KN)-B
       END IF
'
'     STOP EXECUTION IF A ZERO PIVOT IS DETECTED
'
        IF(SS(KN) = 0.00) THEN
          PRINT #2, "STIFFNESS MATRIX IS NOT POSITIVE DEFINITE"
          PRINT #2, USING " PIVOT IS ZERO FOR D.O.F. ####";N
          PRINT #2, USING " PIVOT = #######.#######"; SS(KN)
          STOP
        END IF
'
      NEXT N
      END SUB
'
'**************************************************************************
'*                                REDUCE                                  *
```

```
'***********************************************************************
'
'     PURPOSE: REDUCE REDUCES THE RIGHT-SIDE LOAD VECTOR.
'
'     INPUT : MAXA(*),SS(*),Q(*),NEQ
'     OUTPUT: Q(*)
'
'----------------------
'  IDENTIFIERS
'----------------------
'
'      NEQ          NUMBER OF EQUATIONS (D.O.F.)          INTEGER
'      MAXA(I)      STORES ADDRESSES OF DIAGONAL          INTEGER
'                     ELEMENTS OF K
'                     (SYSTEM STIFFNESS MATRIX) IN
'                     ITS COLUMN VECTOR REPRESENTATION
'      SS(*)        VECTOR REPRESENTATION OF              REAL
'                     SYSTEM STIFFNESS MATRIX
'      Q(NEQ)       APPLIED FORCE DISTRIBUTION            REAL
'                       (UNFACTORED)
'
'

      SUB REDUCE(SS(1),Q(1),MAXA(1),NEQ) STATIC
'
      FOR N=1 TO NEQ
        KL=MAXA(N)+1
        KU=MAXA(N+1)-1
        KH=KU-KL
        IF (KH >= 0) THEN
          K=N
          C=0.00
          FOR KK=KL TO KU
            K=K-1
            C=C+SS(KK)*Q(K)
          NEXT KK
          Q(N)=Q(N)-C
        END IF
      NEXT N
      END SUB
'
'**********************************************************************
'*                                BACSUB                              *
'**********************************************************************
'
'     PURPOSE: BACSUB PERFORMS BACK-SUBSTITUTION TO OBTAIN
'                THE SOLUTION.
'
'     INPUT : NEQ,MAXA(*),SS(*),Q(*)
'     OUTPUT: Q(*)
'
'----------------------
'  IDENTIFIERS
'----------------------
'
'      NEQ          NUMBER OF EQUATIONS (D.O.F.)          INTEGER
'      MAXA(I)      STORES ADDRESSES OF DIAGONAL          INTEGER
'                       ELEMENTS OF K
'                       (SYSTEM STIFFNESS MATRIX) IN
'                       ITS COLUMN VECTOR REPRESENTATION
'      SS(*)          VECTOR REPRESENTATION OF            REAL
'                       SYSTEM STIFFNESS MATRIX
'      Q(NEQ)         APPLIED FORCE DISTRIBUTION          REAL
'                         (UNFACTORED)
'

      SUB BACSUB(SS(1),Q(1),MAXA(1),NEQ) STATIC
'
      FOR N=1 TO NEQ
        K=MAXA(N)
        Q(N)=Q(N)/SS(K)
      NEXT N
      IF (NEQ > 1) THEN
        N=NEQ
        FOR L=2 TO NEQ
          KL=MAXA(N)+1
          KU=MAXA(N+1)-1
          KH=KU-KL
          IF (KH >= 0) THEN
            K=N
            FOR KK=KL TO KU
              K=K-1
              Q(K)=Q(K)-SS(KK)*Q(N)
            NEXT KK
          END IF
          N=N-1
        NEXT L
      END IF
      END SUB
```

APPENDIX C - PASCAL LISTING

```
(*********************************************************************
*   The program SKY initializes variables needed in SKYLIN and       *
* SOLVE and calls routines in proper order.                          *
*                                                                    *
* INPUT: NE,NJ,NEQ,Q,SS,LC                                           *
*                                                                    *
*-----------------------                                             *
*  INPUT REQUIREMENTS                                                *
*-----------------------                                             *
* LIST DIRECTED IN FILE 'SKY.DAT'                                    *
*  1) ENTER NUMBER OF ELEMENTS, NUMBER OF JOINTS, NUBER OF           *
*      EQUATIONS, AND NUMBER OF LOAD CONDITIONS                      *
*            NE,NJ,NEQ,NLC                                           *
*  2) ENTER MEMBER CODE MATRIX ELEMENT BY ELEMENT (i.e., 6 COLUMNS*
*      AND NE ROWS)                                                  *
*          FOR I=1,NE                                                *
*            MCODE((I,J),J=1,6)                                      *
*          CONTINUE                                                  *
*  3) ENTER STIFFNESS MATRIX                                         *
*           SS(1)     SS(2)     SS(3)                                *
*           SS(4)     SS(5)     SS(6)                                *
*           SS(7)     ...       ...                                  *
*            ...      ...       ...                                  *
*                                                                    *
*  4) DO FOR EACH LOAD CONDITION                                     *
*        ENTER FORCE VECTOR Q                                        *
*           Q(1)     Q(2)     Q(3)                                   *
*           Q(4)     Q(5)     Q(6)                                   *
*           Q(7)     ...      ...                                    *
*            ...     ...      ...                                    *
*                                                                    *
*     END DO                                                         *
*                                                                    *
*                                                                    *
*                                                                    *
*                                                                    *
*                                                                    *
*                                                                    *
*                                                                    *
* Passed Parameters:                                                 *
*   KHT               : column height of full tangnet                *
*   LC                : load condition                               *
*   MAXA              : stores addresses of diagonal elements of K   *
*                       (system stiffness matrix) in its column      *
*                       vector representation                        *
*   MCODE             : member code matrix                           *
*                       MCODE[I,J] is the degree of freedom number   *
*                       in the Ith global direction of member J      *
*   NE                : number of elements                           *
*   NEQ               : number of equations                          *
*   NJ              : number of joints                         *
*                     stiffness for degree of freedom I        *
*   Q               : displacement vector                      *
*   SS              : System Stiffness Matrix                  *
*                                                              *
* Attention to the USERS:                                      *
*   Before you run this program, make sure to modify the      *
* constants NE and NEQ to the actual numbers of your own       *
* applications and recompile the program                       *
****************************************************************)
Program SKY(input,output);
  Const MX    =400;             (* To be modified by user *)
        MXNEQ=4000;              (* 3*(MX-1) To be modified by user *)
  Var   i,j,k,l               :integer;
        LC,NLC                :integer;
        LSS                   :integer;
        KHT                   :array [1..MX] of integer;
        MCODE                 :array [1..6,1..MX] of integer;
        NJ                    :integer;
        NE                    :integer;
        NEQ                   :integer;
        SS                    :array [1..MXNEQ] of real;
        Q                     :array [1..MX] of real;
        MAXA                  :array [1..MX] of integer;
        outfile,infile        :text;

   (*****************************************************************
    *   The procedure Timer prints time to screen                   *
    *                                                               *
    *****************************************************************)
    Procedure Timer;
      type
        RegPack = record
                    AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags:integer;
                  end;
      var
        Regs              :RegPack;
        Hour,Min,Sec,Frac :integer;
      begin
        with Regs do
        begin
          AX:=$2C00;
          MsDos(Regs);
          Hour:=hi(CX);
          MIN :=lo(CX);
          Sec :=hi(DX);
          Frac:=lo(DX);
        end;
        writeln('  Time:',Hour:4,':',Min:2,':',Sec:2,'.',Frac:4);
        writeln(outfile,'  Time:',Hour:4,':',Min:2,':',Sec:2,'.',Frac:4);
      end; (* Timer *)

   (*****************************************************************
    *   The program SKYLIN determines KHT using MCODE, and determines *
    * MAXA from KHT                                                 *
  *                                                                    *
  *                              coded                                 *
  *                               by                                   *
  *                           Xudong He                                *
  *                            modified                                *
  *                               by                                   *
  *                       David C. Huxford Jr.                         *
  *                                                                    *
  *                                                                    *
  *                                                                    *
  *                                                                    *
  * INPUT : NE,NEQ,MCODE                                               *
  * OUTPUT: KHT,MAXA,LSS                                               *
  *                                                                    *
  * Passed Parameters:                                                 *
  *   NE                  : number of elements                         *
  *   NEQ                 : number of equations                        *
  *   KHT                 : column height of full tangnet              *
  *                         stiffness for degree of freedom I          *
  *   MAXA                : stores addresses of diagonal elements of K *
  *                         (system stiffness matrix) in its column    *
  *                         vector representation                      *
  *   MCODE               : member code matrix                         *
  *                         MCODE[I,J] is the degree of freedom number *
  *                         in the Ith global direction of member J    *
  *                                                                    *
  * Attention to the USERS:                                            *
  *   Before you run this program, make sure to modify the            *
  * constants NE and NEQ to the actual numbers of your own             *
  * applications and recompile the program                             *
  *********************************************************************)
procedure SKYLIN;
  Var k,min                 :integer;
begin (*SKYLIN*)
  (*Initialization*)
  for i:=1 to NEQ do
  begin
    KHT[i]:=0;maxa[i]:=0;
  end;
  MAXA[NEQ+1]:=0;

  (* Generate KHT *)
  for i:=1 to NE do
  begin
    j:=1;
    while (MCODE[j,i]=0) and (j<6) do j:=j+1;
    min:=MCODE[j,i]; j:=j+1;
    for l:=j to 6 do
    begin
      k:=MCODE[l,i];
      if (k<>0) then
       if KHT[k] < (k-min) then KHT[k]:=k-min;
    end;
  end;

  (* generate MAXA *)
  MAXA[1]:=1;
```

```
   for i:=1 to NEQ do MAXA[i+1]:=MAXA[i]+KHT[i]+1;
   LSS:=MAXA[NEQ+1]-1;

   (* Print the OUTPUT *)
   writeln(outfile);
   writeln(outfile,'          I        KHT(I)       MAXA(I)');
   for i:=1 to NEQ do
     writeln(outfile,i:10,KHT[i]:10,MAXA[i]:10);

   writeln(outfile,(NEQ+1):10,MAXA[NEQ+1]:20);
   writeln(outfile);
   writeln(outfile,'     LSS=                ',LSS:5);
 end; (* SKYLIN *)

(*********************************************************************
 *   The procedure QOUT echos Q                                      *
 *                                                                   *
 * OUTPUT:                                                           *
 *   Q                  : Displacements                              *
 *********************************************************************)
   Procedure QOUT;
     Var  NUM                         :integer;
   begin
     NUM:=trunc(NEQ/3);
     for i:=1 to NUM do
     begin
       writeln(outfile,'Q(',(i*3-2):3,')= ',Q[i*3-2]:14:7,
                       ' Q(',(i*3-1):3,')= ',Q[i*3-1]:14:7,
                       ' Q(',(i*3):3,')= ',Q[i*3]:14:7);
     end;
     NUM:=NEQ-(NUM*3);
     if (NUM <> 0) then
     begin
       if (NUM = 1) then
       begin
         writeln(outfile,'Q(',(NEQ-NUM+1):3,')= ',Q[NEQ-NUM+1]:14:7);
       end;
       if (NUM = 2) then
       begin
         writeln(outfile,'Q(',(NEQ-NUM+1):3,')= ',Q[NEQ-NUM+1]:14:7,
                         ' Q(',(NEQ-NUM+2):3,')= ',Q[NEQ-NUM+2]:14:7);
       end
     end
   end; (* QOUT *)

(*********************************************************************
 *   The program SOLVE determines the solution to the system         *
 * equations by ACTIVE COLUMN SOLUTION (SKYLINE REDUCTION METHOD)    *
 * based on the subroutine COLSOL (BATHE pp.448-449)                 *
 *                                                                   *
 *                                                                   *
 *                           coded                                   *
 *                            by                                     *
 *                         Xudong He                                 *
 *                          modified                                 *
 *                            by                                     *
 *                    David C. Huxford Jr.                           *
*                                                                    *
*                                                                    *
* INPUT :                                                            *
*   NEQ               : number of equations            integer       *
*   LSS               : number of elements in SS       integer       *
*   LC                : load condition                               *
*   MAXA              : stores addresses of diagonal elements of K   *
*                       (system stiffness matrix) in its column      *
*                       vector representation                        *
*                                              one dimensional array *
*   SS                : vector representation of System Stiffness    *
*                       matrix                 one dimensional array *
*   Q                 : right-hand side   load vecotor               *
*                                              one dimensional array *
*                                                                    *
* OUTPUT:                                                            *
*   Q                 : the solution                                 *
*                                                                    *
*                                                                    *
*                                                                    *
*                                                                    *
* Attention to the USERS:                                            *
*   Before you run this program, make sure to modify the             *
* constants NEQ and LSS to the actual numbers of your own            *
* applications and recompile the program                             *
**********************************************************************)
Procedure SOLVE;
  Var i,j,k,l,NUM                 : integer; (* loop index variables *)
      kn,kl,ku,kh,ki,klt,ic,kk : integer; (* temporary variables *)
      r,s                         : real;    (* temporary variables *)
      pivot                       : boolean; (* pivot control variable *)

   (**************************************************************
    *   The procedure FACTOR performs the LDU factorization of *
    * the Stiffness Matrix.                                     *
    *   It uses the arrays: MAXA and SS and changes SS to       *
    * factorized form                                           *
    *                                                           *
    *                          coded                            *
    *                           by                              *
    *                       Xudong He                           *
    *                        modified                           *
    *                           by                              *
    *                  David C. Huxford Jr.                     *
    *                                                           *
    ***************************************************************)
   Procedure FACTOR;
   begin (* FACTOR *)
     i:=1;
     while (i <= NEQ) and pivot do
     begin
       (* Calculate the height of column i *)
       kn:=MAXA[i];
       kl:=kn+1;
       ku:=MAXA[i+1]-1;
       kh:=ku-kl;
      if kh > 0 then
      begin
        k:=i-kh;
        ic:=0;
        klt:=ku;
        for j:=1 to kh do
        begin
          ic:=ic+1;
          klt:=klt-1;
          ki:=MAXA[k];
          if (MAXA[k+1]-ki-1) > 0 then
          begin
            if (MAXA[k+1]-ki-1) > ic then kk:=ic
            else kk:=MAXA[k+1]-ki-1;
            r:=0.0;
            for l:=1 to kk do r:= r+SS[ki+l]*SS[klt+l];
            SS[klt]:=SS[klt] - r;
          end;
          k:=k+1;
        end;
      end;

      if kh >= 0 then
      begin
        k:=i;
        r:=0.0;
        for kk:=kl to ku do
        begin
          k:=k-1;
          ki:=MAXA[k];
          s:=SS[kk]/SS[ki];
          r:=r+s*SS[kk];
          SS[kk]:= s;
        end;
        SS[kn]:=SS[kn]-r
      end;

      (* Stop Execution if a Zero Pivot is Detected *)
      if (abs(SS[kn]) <= 0.0000000001) then
      begin
        pivot:=false;
        writeln(outfile);
        writeln (outfile,'Stiffness Matrix is not Positive Definite');
        writeln (outfile,'Pivot is Zero for D.O.F. ',i:5,
                         ' PIVOT= ',SS[kn]:12:7);
      end;
      i:=i+1
    end;
  end; (* FACTOR *)

  (************************************************************
   *   The procedure REDUCE reduces the right-side load       *
   * vector.                                                  *
   *   It uses the arrays: MAXA, SS, Q and modifies Q.        *
   *                                                          *
   *                          coded                           *
   *                           by                             *
```

```
 *                          Xudong He                           *
 *                           modified                           *
 *                              by                              *
 *                     David C. Huxford Jr.                     *
 *                                                              *
 ***************************************************************)
Procedure REDUCE;
begin
  for i:=1 to NEQ do
  begin
    kl:=MAXA[i]+1;
    ku:=MAXA[i+1]-1;
    kh:=ku-kl;
    if (kh >= 0) then
    begin
      k:=i;
      r:=0.0;
      for kk:=kl to ku do
      begin
        k:=k-1;
        r:=r+SS[kk]*Q[k]
      end;
      Q[i]:=Q[i]-r
    end
  end
end; (* REDUCE *)

(***************************************************************
 *   The procedure BACSUB performs Back-Substitution to         *
 * obtain the solution.                                         *
 *   It uses the arrays: MAXA, SS Q and modifies Q              *
 *                                                              *
 *                            coded                             *
 *                              by                              *
 *                          Xudong He                           *
 *                           modified                           *
 *                              by                              *
 *                     David C. Huxford Jr.                     *
 *                                                              *
 ***************************************************************)
Procedure BACSUB;
begin (* BACSUB *)
  for i:=1 to NEQ do
  begin
    k:=MAXA[i];
    Q[i]:=Q[i]/SS[k]
  end;
  if NEQ > 1 then
  begin
    i:=NEQ;
    for j:=2 to NEQ do
    begin
      kl:=MAXA[i]+1;
      ku:=MAXA[i+1]-1;
      kh:=ku-kl;
      if (kh >= 0) then
      begin
```

```
              k:=i;
              for kk:=kl to ku do
              begin
                k:=k-1;
                Q[k]:=Q[k]-SS[kk]*Q[i]
              end;
            end;
            i:=i-1
          end
        end
      end; (* BACSUB *)

  begin (* SOLVE *)
    pivot:=true;
    writeln(outfile);
    writeln(outfile,'Computational Time Begin-End');
    writeln('Computational Time Begin-End');
    Timer;
    if (LC =1) then FACTOR;
    if pivot then
    begin
      REDUCE;
      BACSUB;

      Timer;
      (* Print out the Results *)
      writeln(outfile);
      writeln(outfile);
      writeln(outfile,'DISPLACEMENTS');
      QOUT;
    end
  end; (* SOLVE *)

  (*********************************************************************
   *   The procedure IN initializes MCODE for use in SKYLIN.           *
   *                                                                   *
   * INPUT :                                                           *
   *   NE              : number of elements             integer        *
   *                                                                   *
   * OUTPUT:                                                           *
   *   MCODE           : Member Code Matrix                            *
   *                                                                   *
   *                                                                   *
   *********************************************************************)
  Procedure INPUT;
  begin (* INPUT *)
    writeln(outfile,'MCODE Transposed');
    for i:=1 to NE do
    begin
      readln(infile,MCODE[1,i],MCODE[2,i],MCODE[3,i],MCODE[4,i],
                    MCODE[5,i],MCODE[6,i]);
      writeln(outfile,MCODE[1,i]:5,MCODE[2,i]:5,MCODE[3,i]:5,MCODE[4,i]:5,
                      MCODE[5,i]:5,MCODE[5,i]:5);
    end
  end; (* INPUT *)

  (*********************************************************************
 *   The procedure SSINPUT reads and echos SS for use in SOLVE     *
 *                                                                 *
 * INPUT :                                                         *
 *   NEQ               : number of elements            integer    *
 *   LSS               : number of elements in SS      integer    *
 *                                                                 *
 * OUTPUT:                                                         *
 *   SS                : System Stiffnes matrix                    *
 *   Q                 : Displacements                             *
 ******************************************************************)
Procedure SSINPUT;
  Var NUM                      : integer;
begin (* QINPUT *)
  writeln(outfile);
  writeln(outfile);
  writeln(outfile,'SS MATRIX');
  NUM:=trunc(LSS/3);
  for i:=1 to NUM do
  begin
    readln(infile,SS[i*3-2],SS[i*3-1],SS[i*3]);
    writeln(outfile,'SS(',(i*3-2):3,')= ',SS[i*3-2]:13:6,
                    ' SS(',(i*3-1):3,')= ',SS[i*3-1]:13:6,
                    ' SS(',(i*3):3,')= ',SS[i*3]:13:6)
  end;
  NUM:=LSS-(NUM*3);
  if (NUM <> 0) then
  begin
    if (NUM = 1) then
    begin
      readln(infile,SS[LSS-NUM+1]);
      writeln(outfile,'SS(',(LSS-NUM+1):3,')= ',SS[LSS-NUM+1]:13:6);
    end;
    if (NUM = 2) then
    begin
      readln(infile,SS[LSS-NUM+1],SS[LSS-NUM+2]);
      writeln(outfile,'SS(',(LSS-NUM+1):3,')= ',SS[LSS-NUM+1]:13:6,
                      ' SS(',(LSS-NUM+2):3,')= ',SS[LSS-NUM+2]:13:6);
    end;
  end;
end; (* SSINPUT *)
(*****************************************************************
 *   The procedure QINPUT reads and echos Q for use in SOLVE      *
 *                                                                 *
 * INPUT :                                                         *
 *   NEQ               : number of elements            integer    *
 *   LSS               : number of elements in SS      integer    *
 *                                                                 *
 * OUTPUT:                                                         *
 *   SS                : System Stiffnes matrix                    *
 *   Q                 : Displacements                             *
 ******************************************************************)
Procedure QINPUT;
  Var NUM                      : integer;
begin (* QINPUT *)
  writeln(outfile);
  writeln(outfile);
  writeln(outfile,'FORCES');
```

```
      NUM:=trunc(NEQ/3);
      for i:=1 to NUM do
      begin
        readln(infile,Q[i*3-2],Q[i*3-1],Q[i*3]);
      end;
      NUM:=NEQ-(NUM*3);
      if (NUM <> 0) then
      begin
        if (NUM = 1) then
        begin
          readln(infile,Q[NEQ-NUM+1]);
        end;
        if (NUM = 2) then
        begin
          readln(infile,Q[NEQ-NUM+1],Q[NEQ-NUM+2]);
        end
      end;
    QOUT;
    end; (* QINPUT *)

begin (* SKY *)
  assign (infile,'sky.dat'); reset(infile);
  assign (outfile,'sky.out'); rewrite(outfile);
  readln(infile,NE,NJ,NEQ,NLC);
  writeln(outfile,'NE= ',NE:5);
  writeln(outfile,'NJ= ',NJ:5);
  writeln(outfile,'NEQ= ',NEQ:5);
  writeln(outfile,'LC= ',NLC:5);
  writeln(outfile);
  writeln(outfile);
  for i:=1 to MXNEQ do
  begin
    SS[i]:=0.0;
  end;
  INPUT;
  SKYLIN;
  if (LSS < MXNEQ) then
  begin
    SSINPUT;
    for LC:=1 to NLC do
    begin
      for i:=1 to MX do
      begin
       Q[i]:=0.0;
      end;
      writeln(outfile);
      writeln(outfile);
      writeln('LOAD CONDITION ',LC:3);
      writeln(outfile,'LOAD CONDITION ',LC:3);
      writeln(outfile);
      QINPUT;
      SOLVE;
    end;
    close(infile);
    close(outfile);
  end
  else
```

```
     writeln('ERROR MESSAGE: INCREASE MEMORY');
 END. (* SKY *)
```

APPENDIX D - C LISTING

```
/*********************************************************************
*   The program SKY initializes variables needed in SKYLIN and      *
* SOLVE and calls routines in proper order.                          *
*                                                                    *
*                                                                    *
*                                                                    *
*------------------------                                            *
*  INPUT REQUIREMENTS                                                *
*------------------------                                            *
* LIST DIRECTED IN FILE 'SKY.DAT'                                    *
*  1) ENTER NUMBER OF ELEMENTS, NUMBER OF JOINTS, NUBER OF           *
*      EQUATIONS AND NUMBER OF LOAD CONDITIONS                       *
*            NE,NJ,NEQ,NLC                                           *
*  2) ENTER MEMBER CODE MATRIX ELEMENT BY ELEMENT (i.e., 6 COLUMNS*
*      AND NE ROWS)                                                  *
*          FOR I=1,NE                                                *
*            MCODE((I,J),J=1,6)                                      *
*          CONTINUE                                                  *
*  3) ENTER STIFFNESS MATRIX                                         *
*           SS(1)     SS(2)     SS(3)                                *
*           SS(4)     SS(5)     SS(6)                                *
*           SS(7)     ...       ...                                  *
*            ...      ...       ...                                  *
*                                                                    *
*  4) DO FOR EACH LOAD CONDITION                                     *
*         ENTER FORCE VECTOR Q                                       *
*           Q(1)      Q(2)      Q(3)                                 *
*           Q(4)      Q(5)      Q(6)                                 *
*           Q(7)      ...       ...                                  *
*            ...      ...       ...                                  *
*                                                                    *
*     END DO                                                         *
*                                                                    *
*                                                                    *
*                                                                    *
* Passed Parameters:                                                 *
*   KHT              : column height of full tangnet                 *
*   LC               : load condition                                *
*   MAXA             : stores addresses of diagonal elements of K    *
*                      (system stiffness matrix) in its column       *
*                      vector representation                         *
*   MCODE            : member code matrix                            *
*                      MCODE[I,J] is the degree of freedom number    *
*                      in the Ith global direction of member J       *
*   NE               : number of elements                            *
*   NEQ              : number of equations                           *
*   NJ               : number of joints                              *
*                      stiffness for degree of freedom I             *
*   Q                : displacement vector                           *
*   SS               : System Stiffness Matrix                       *
*                                                                        *
* Attention to the USERS:                                                *
*   Before you run this program, make sure to modify the                *
* constants NE and NEQ to the actual numbers of your own                 *
* applications and recompile the program                                 *
*************************************************************************/
#include <stdio.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <io.h>
#include <stdlib.h>
#include <time.h>
#define MX    400                                 /* To be modified by user */
#define MXNEQ 4000                                /* To be modified by user */
        int                 i,j,k,l;              /* loop index variables */
        int                 pivot;                /* pivot control variable */
        int                 LC,NLC;
        int                 LSS;
        int                 KHT [MX+1];
        int                 MCODE [7][MX+1];
        int                 NJ;
        int                 NE;
        int                 NEQ;
        int                 MAXA [MX+1];
        float               SS [MXNEQ+1];
        float               Q [MX+1];
        FILE                *in,*out;             /* file pointers */


   /*********************************************************************
    *   The program SKYLIN determines KHT using MCODE, and determines *
    * MAXA from KHT                                                     *
    *                                                                   *
    *                         coded                                     *
    *                          by                                       *
    *                       Xudong He                                   *
    *                       Yingjia Ding                                *
    *                        modified                                   *
    *                          by                                       *
    *                   David C. Huxford Jr.                            *
    *                                                                   *
    *                                                                   *
    *                                                                   *
    *                                                                   *
    * INPUT : NE,NEQ,MCODE                                              *
    * OUTPUT: KHT,MAXA,LSS                                              *
    *                                                                   *
    * Passed Parameters:                                                *
    *   NE            : number of elements                              *
    *   NEQ           : number of equations                             *
    *   KHT           : column height of full tangnet                   *
    *                   stiffness for degree of freedom I               *
    *   MAXA          : stores addresses of diagonal elements of K     *
    *                   (system stiffness matrix) in its column         *
    *                   vector representation                           *
    *   MCODE         : member code matrix                              *
```

```
 *                         MCODE[I,J] is the degree of freedom number  *
 *                         in the Ith global direction of member J     *
 *                                                                     *
 * Attention to the USERS:                                             *
 *   Before you run this program, make sure to modify the              *
 * constants NE and NEQ to the actual numbers of your own              *
 * applications and recompile the program                              *
 **********************************************************************/
int SKYLIN ()
{     /*SKYLIN*/
  int k,min;

  /*Initialization*/
  for (i=0;i<=NEQ;i++)
  {
    KHT[i]=0;MAXA[i]=0;
  }
  MAXA[NEQ+1]=0;

  /* Generate KHT */
  for (i=1;i<=NE;i++)
  {
    j=1;
    while ((MCODE[j][i]==0) && (j<6)) j=j+1;
    min=MCODE[j][i]; j=j+1;
    for (l=j;l<=6;l++)
    {
      k=MCODE[l][i];
      if (k != 0)
       if (KHT[k] < (k-min)) KHT[k]=k-min;
    }
  }

  /* generate MAXA */
  MAXA[1]=1;
  for (i=1;i<=NEQ;i++)  MAXA[i+1]=MAXA[i]+KHT[i]+1;
  LSS=MAXA[NEQ+1]-1;

  /* Print the OUTPUT */
  fprintf(out,"\n");
  fprintf(out,"        I       KHT(I)       MAXA(I)\n");
  for(i=1;i<=NEQ;i++)
  {
    fprintf(out,"        %d           %d             %d\n",i,
                                                   KHT[i],MAXA[i]);
  }
  fprintf(out,"        %d                          %d\n",(NEQ+1),
                                                  MAXA[NEQ+1]);
  fprintf(out,"\n");
  fprintf(out,"        LSS=                        %d\n",LSS);
return;
} /* SKYLIN */

/**********************************************************************
 *   The procedure QOUT echos Q                                        *
 *                                                                     *
 * OUTPUT:                                                             *
*   Q                    : Displacements                                *
***********************************************************************/
 float QOUT ()
 {
   int NUM;

   NUM=(NEQ/3);
   for (i=1;i<=NUM;i++)
   {
     fprintf(out,"Q(%4d)= %14.7f  Q(%4d)= %14.7f  Q(%4d)= %14.7f\n",
             (i*3-2),Q[i*3-2],(i*3-1),Q[i*3-1],(i*3),Q[i*3]);
   }
   NUM=NEQ-(NUM*3);
   if (NUM != 0)
   {
     if (NUM == 1)
     {
       fprintf(out,"Q(%4d)= %14.7f\n",(NEQ-NUM+1),Q[NEQ-NUM+1]);
     }
     if (NUM == 2)
     {
       fprintf(out,"Q(%4d)= %14.7f  Q(%4d)= %14.7f\n",(NEQ-
NUM+1),Q[NEQ-NUM+1],
                                        (NEQ-NUM+2),Q[NEQ-NUM+2]);
     }
   }
 return;
 } /* QOUT */

/***********************************************************************
*   The program SOLVE determines the solution to the system             *
* equations by ACTIVE COLUMN SOLUTION (SKYLINE REDUCTION METHOD)        *
* based on the subroutine COLSOL (BATHE pp.448-449)                     *
*                                                                       *
*                                                                       *
*                               coded                                   *
*                                by                                     *
*                             Xudong He                                 *
*                            Yingjia Ding                               *
*                              modified                                 *
*                                by                                     *
*                         David C. Huxford Jr.                          *
*                                                                       *
*                                                                       *
* INPUT :                                                               *
*   NEQ              : number of equations            integer           *
*   LSS              : number of elements in SS       integer           *
*   LC               : load condition                                   *
*   MAXA             : stores addresses of diagonal elements of K       *
*                      (system stiffness matrix) in its column          *
*                      vector representation                            *
*                                           one dimensional array       *
*   SS               : vector representation of System Stiffness        *
*                      matrix               one dimensional array       *
*   Q                : right-hand side   load vecotor                   *
*                                           one dimensional array       *
*                                                                       *
* OUTPUT:                                                        *
*   Q                : the solution                              *
*                                                                *
*                                                                *
*                                                                *
*                                                                *
* Attention to the USERS:                                        *
*   Before you run this program, make sure to modify the         *
* constants NEQ and LSS to the actual numbers of your own        *
* applications and recompile the program                         *
*****************************************************************/


/*************************************************************
 *   The procedure FACTOR performs the LDU factorization of *
 * the Stiffness Matrix.                                     *
 *   It uses the arrays: MAXA and SS and changes SS to       *
 * factorized form                                           *
 *                                                           *
 *                         coded                             *
 *                          by                               *
 *                     Yingjia Ding                          *
 *                      Xudong He                            *
 *                      modified                             *
 *                          by                               *
 *                  David C. Huxford Jr.                     *
 *                                                           *
 *************************************************************/
float FACTOR ()
{    /* FACTOR */
  int kn,kl,ku,kh,ki,klt,ic,kk; /* temporary variables */
  float r,s;                    /* temporary variables */

  i=1;
  while ((i <= NEQ) && (pivot==1))
  {
    /* Calculate the height of column i */
    kn=MAXA[i];
    kl=kn+1;
    ku=MAXA[i+1]-1;
    kh=ku-kl;

    if (kh > 0)
    {
      k=i-kh;
      ic=0;
      klt=ku;
      for(j=1;j<=kh;j++)
      {
        ic=ic+1;
        klt=klt-1;
        ki=MAXA[k];
        if ((MAXA[k+1]-ki-1) > 0)
        {
          if ((MAXA[k+1]-ki-1) > ic) kk=ic;
          else kk=MAXA[k+1]-ki-1;
          r=0.0;
          for(l=1;l<=kk;l++)  r= r+SS[ki+l]*SS[klt+l];
          SS[klt]=SS[klt] - r;
        }
        k=k+1;
      }
    }

    if (kh >= 0)
    {
      k=i;
      r=0.0;
      for(kk=kl;kk<=ku;kk++)
      {
        k=k-1;
        ki=MAXA[k];
        s=SS[kk]/SS[ki];
        r=r+s*SS[kk];
        SS[kk]= s;
      }
      SS[kn]=SS[kn]-r;
    }

    /* Stop Execution if a Zero Pivot is Detected */
    if (fabs(SS[kn]) <= 0.0000000001)
    {
      pivot=0;
      fprintf(out,"/n");
      fprintf(out,"Stiffness Matrix is not Positive Definite\n");
      fprintf(out,"Pivot is Zero for D.O.F. %d PIVOT= %f\n",
                                                 i,SS[kn]);
    }
    i++;
  }
return;
} /* FACTOR */

/************************************************************
 *   The procedure REDUCE reduces the right-side load       *
 * vector.                                                  *
 *   It uses the arrays: MAXA, SS, Q and modifies Q.        *
 *                                                          *
 *                          coded                           *
 *                           by                             *
 *                        Xudong He                         *
 *                       Yingjia Ding                       *
 *                         modified                         *
 *                           by                             *
 *                     David C. Huxford Jr.                 *
 *                                                          *
 ************************************************************/
float REDUCE ()
{
  int   kn,kl,ku,kh,ki,klt,ic,kk; /* temporary variables */
  float r;                        /* temporary variables */

  for (i=1;i<=NEQ;i++)
  {
    kl=MAXA[i]+1;
    ku=MAXA[i+1]-1;
    kh=ku-kl;
    if (kh >= 0)
    {
      k=i;
      r=0.0;
      for (kk=kl;kk<=ku;kk++)
      {
        k=k-1;
        r=r+SS[kk]*Q[k];
      }
      Q[i]=Q[i]-r;
    }
  }
return;
} /* REDUCE */

/*************************************************************
 *   The procedure BACSUB performs Back-Substitution to     *
 * obtain the solution.                                      *
 *   It uses the arrays: MAXA, SS Q and modifies Q           *
 *                                                           *
 *                            coded                          *
 *                             by                            *
 *                         Xudong He                         *
 *                        Yingjia Ding                       *
 *                          modified                         *
 *                             by                            *
 *                     David C. Huxford Jr.                  *
 *                                                           *
 *************************************************************/
float BACSUB ()
{   /* BACSUB */
  int   kn,kl,ku,kh,ki,klt,ic,kk; /* temporary variables */

  for (i=1;i<=NEQ;i++)
  {
    k=MAXA[i];
    Q[i]=Q[i]/SS[k];
  }
  if (NEQ > 1)
  {
    i=NEQ;
    for (j=2;j<=NEQ;j++)
    {
      kl=MAXA[i]+1;
      ku=MAXA[i+1]-1;
      kh=ku-kl;
      if (kh >= 0)
      {
        k=i;
        for(kk=kl;kk<=ku;kk++)
        {
          k=k-1;
          Q[k]=Q[k]-SS[kk]*Q[i];
        }
```

```
            }
            i=i-1;
          }
        }
      return;
      } /* BACSUB */

float SOLVE ()
{  /* SOLVE */
  time_t start,finish;

  pivot=1;
  time(&start);
  if (LC ==1 ) FACTOR();
  if (pivot == 1)
  {
    REDUCE();
    BACSUB();
    time(&finish);
    printf("  Start  %s",ctime(&start));
    printf("  End    %s",ctime(&finish));
    printf("  Time elapsed in Solution Process: %f Seconds\n",
            difftime(finish,start));
    fprintf(out,"\n  Start  %s",ctime(&start));
    fprintf(out,"  End    %s",ctime(&finish));
    fprintf(out,"  Time elapsed in Solution Process: %f Seconds\n",
            difftime(finish,start));
    /* Print out the Results */
    fprintf(out,"\nDISPLACEMENTS\n");
    QOUT();
  }
return;
} /* SOLVE */

/*********************************************************************
 *   The procedure IN initializes MCODE for use in SKYLIN.          *
 *                                                                  *
 * INPUT :                                                          *
 *   NE               : number of elements              integer     *
 *                                                                  *
 * OUTPUT:                                                          *
 *   MCODE            : Member Code Matrix                          *
 *                                                                  *
 *                                                                  *
 *********************************************************************/
float INPUT ()
{    /* INPUT */
  fprintf(out,"MCODE Transposed\n");
  for(i=1;i<=NE;i++)
  {
    fscanf(in,"%d %d %d %d %d %d",&MCODE[1][i],&MCODE[2][i],
                              &MCODE[3][i],&MCODE[4][i],
                              &MCODE[5][i],&MCODE[6][i]);
    fprintf(out,"%4d %4d %4d %4d %4d %4d\n",
                              MCODE[1][i],MCODE[2][i],
                              MCODE[3][i],MCODE[4][i],
                              MCODE[5][i],MCODE[6][i]);
  }
return;
} /* INPUT */

/**********************************************************************
 *   The procedure SSINPUT reads and echos SS for use in SOLVE.      *
 *                                                                    *
 * INPUT :                                                            *
 *   LSS              : number of elements in SS        integer      *
 *                                                                    *
 * OUTPUT:                                                            *
 *   SS               : System Stiffnes matrix                        *
 **********************************************************************/
float SSINPUT ()
{   /* SSINPUT */
  int NUM;

  fprintf(out,"\n");
  fprintf(out,"\n");
  fprintf(out,"SS MATRIX\n");
  NUM=(LSS/3);
  for(i=1;i<=NUM;i++)
  {
    fscanf(in,"%f %f %f",&SS[i*3-2],&SS[i*3-1],&SS[i*3]);
    fprintf(out,"SS(%4d)= %14.7f  SS(%4d)= %14.7f  SS(%4d)= %14.7f\n",
                                                  (i*3-2),SS[i*3-2],
                                                  (i*3-1),SS[i*3-1],
                                                  (i*3),SS[i*3]);
  }
  NUM=LSS-(NUM*3);
  if (NUM != 0)
  {
    if (NUM == 1)
    {
      fscanf(in,"%f",&SS[LSS-NUM+1]);
      fprintf(out,"SS(%4d)= %14.7f\n",(LSS-NUM+1),SS[LSS-NUM+1]);
    }
    if (NUM == 2)
    {
      fscanf(in,"%f %f",&SS[LSS-NUM+1],&SS[LSS-NUM+2]);
      fprintf(out,"SS(%4d)= %14.7f  SS(%4d)= %14.7f\n",
                                         (LSS-NUM+1),SS[LSS-NUM+1],
                                         (LSS-NUM+2),SS[LSS-NUM+2]);
    }
  }
return;
} /* SSINPUT */
/**********************************************************************
 *   The procedure QINPUT reads and echos Q for use in SOLVE.        *
 *                                                                    *
 * INPUT :                                                            *
 *   NEQ              : number of elements              integer      *
 *                                                                    *
 * OUTPUT:                                                            *
 *   Q                : Displacements                                 *
 **********************************************************************/
float QINPUT ()
```

```
    {    /* QINPUT */
      int NUM;

      fprintf(out,"\n");
      fprintf(out,"\n");
      fprintf(out,"FORCES\n");

      NUM=(NEQ/3);
      for(i=1;i<=NUM;i++)
      {
        fscanf(in,"%f %f %f",&Q[i*3-2],&Q[i*3-1],&Q[i*3]);
      }
      NUM=NEQ-(NUM*3);
      if (NUM != 0)
      {
        if (NUM == 1)
        {
          fscanf(in,"%f",&Q[NEQ-NUM+1]);
        }
        if (NUM == 2)
        {
          fscanf(in,"%f %f",&Q[NEQ-NUM+1],&Q[NEQ-NUM+2]);
        }
      }
      QOUT ();
    return;
    } /* QINPUT */

main ()   /* SKY */
{    /* SKY */
  in=fopen("sky.dat","r");
  out=fopen("sky.out","w");
  fscanf(in,"%d %d %d %d",&NE,&NJ,&NEQ,&NLC);
  fprintf(out,"NE= %d\n",NE);
  fprintf(out,"NJ= %d\n",NJ);
  fprintf(out,"NEQ= %d\n",NEQ);
  fprintf(out,"LC= %d\n",NLC);
  fprintf(out,"\n");
  fprintf(out,"\n");
  for(i=1;i<=MXNEQ;i++)
  {
    SS[i]=0.0;
  }
  INPUT ();
  SKYLIN ();
  SSINPUT ();
  if (LSS < MXNEQ)
  {
     for(LC=1;LC<=NLC;LC++)
     {
       for(i=1;i<=MX;i++)
       {
         Q[i]=0.0;
       }
       printf(" LOAD CONDITION %d\n",LC);
       fprintf(out,"\n\n LOAD CONDITION %d\n",LC);
       QINPUT ();
```

```
          SOLVE ();
          close(in);
          close(out);
        }
    }
    else printf("ERROR MESSAGE: INCREASE MEMORY\n");
  } /* SKY */
```

APPENDIX E - QUICKBASIC OUTPUT

```
NE=  144
NJ=   85
NEQ= 240
LC=    2


MCODE TRANSPOSED
   1    2    3    4    5    6
   4    5    6    7    8    9
   7    8    9   10   11   12
  10   11   12   13   14   15
   1    2    3   16   17   18
   4    5    6   19   20   21
   7    8    9   22   23   24
  10   11   12   25   26   27
  13   14   15   28   29   30
  16   17   18   19   20   21
  19   20   21   22   23   24
  22   23   24   25   26   27
  25   26   27   28   29   30
  16   17   18   31   32   33
  19   20   21   34   35   36
  22   23   24   37   38   39
  25   26   27   40   41   42
  28   29   30   43   44   45
  31   32   33   34   35   36
  34   35   36   37   38   39
  37   38   39   40   41   42
  40   41   42   43   44   45
  31   32   33   46   47   48
  34   35   36   49   50   51
  37   38   39   52   53   54
  40   41   42   55   56   57
  43   44   45   58   59   60
  46   47   48   49   50   51
  49   50   51   52   53   54
  52   53   54   55   56   57
  55   56   57   58   59   60
  46   47   48   61   62   63
  49   50   51   64   65   66
  52   53   54   67   68   69
  55   56   57   70   71   72
  58   59   60   73   74   75
  61   62   63   64   65   66
  64   65   66   67   68   69
  67   68   69   70   71   72
  70   71   72   73   74   75
  61   62   63   76   77   78
  64   65   66   79   80   81
  67   68   69   82   83   84
  70   71   72   85   86   87
  73   74   75   88   89   90
  76   77   78   79   80   81
  79   80   81   82   83   84
  82   83   84   85   86   87
  85   86   87   88   89   90
  76   77   78   91   92   93
  79   80   81   94   95   96
```

82	83	84	97	98	99
85	86	87	100	101	102
88	89	90	103	104	105
91	92	93	94	95	96
94	95	96	97	98	99
97	98	99	100	101	102
100	101	102	103	104	105
91	92	93	106	107	108
94	95	96	109	110	111
97	98	99	112	113	114
100	101	102	115	116	117
103	104	105	118	119	120
106	107	108	109	110	111
109	110	111	112	113	114
112	113	114	115	116	117
115	116	117	118	119	120
106	107	108	121	122	123
109	110	111	124	125	126
112	113	114	127	128	129
115	116	117	130	131	132
118	119	120	133	134	135
121	122	123	124	125	126
124	125	126	127	128	129
127	128	129	130	131	132
130	131	132	133	134	135
121	122	123	136	137	138
124	125	126	139	140	141
127	128	129	142	143	144
130	131	132	145	146	147
133	134	135	148	149	150
136	137	138	139	140	141
139	140	141	142	143	144
142	143	144	145	146	147
145	146	147	148	149	150
136	137	138	151	152	153
139	140	141	154	155	156
142	143	144	157	158	159
145	146	147	160	161	162
148	149	150	163	164	165
151	152	153	154	155	156
154	155	156	157	158	159
157	158	159	160	161	162
160	161	162	163	164	165
151	152	153	166	167	168
154	155	156	169	170	171
157	158	159	172	173	174
160	161	162	175	176	177
163	164	165	178	179	180
166	167	168	169	170	171
169	170	171	172	173	174
172	173	174	175	176	177
175	176	177	178	179	180
166	167	168	181	182	183
169	170	171	184	185	186
172	173	174	187	188	189
175	176	177	190	191	192
178	179	180	193	194	195
181	182	183	184	185	186
184	185	186	187	188	189
187	188	189	190	191	192
190	191	192	193	194	195
181	182	183	196	197	198
184	185	186	199	200	201

```
187 188 189 202 203 204
190 191 192 205 206 207
193 194 195 208 209 210
196 197 198 199 200 201
199 200 201 202 203 204
202 203 204 205 206 207
205 206 207 208 209 210
196 197 198 211 212 213
199 200 201 214 215 216
202 203 204 217 218 219
205 206 207 220 221 222
208 209 210 223 224 225
211 212 213 214 215 216
214 215 216 217 218 219
217 218 219 220 221 222
220 221 222 223 224 225
211 212 213 226 227 228
214 215 216 229 230 231
217 218 219 232 233 234
220 221 222 235 236 237
223 224 225 238 239 240
226 227 228 229 230 231
229 230 231 232 233 234
232 233 234 235 236 237
235 236 237 238 239 240
226 227 228   0   0   0
229 230 231   0   0   0
232 233 234   0   0   0
235 236 237   0   0   0
238 239 240   0   0   0
```

```
         I            KHT(I)          MAXA(I)
         1              0                1
         2              1                2
         3              2                4
         4              3                7
         5              4               11
         6              5               16
         7              3               22
         8              4               26
         9              5               31
        10              3               37
        11              4               41
        12              5               46
        13              3               52
        14              4               56
        15              5               61
        16             15               67
        17             16               83
        18             17              100
        19             15              118
        20             16              134
        21             17              151
        22             15              169
        23             16              185
        24             17              202
        25             15              220
        26             16              236
        27             17              253
        28             15              271
        29             16              287
        30             17              304
```

[illegible]	[illegible]	[illegible]
31	15	322
32	16	338
33	17	355
34	15	373
35	16	389
36	17	406
37	15	424
38	16	440
39	17	457
40	15	475
41	16	491
42	17	508
43	15	526
44	16	542
45	17	559
46	15	577
47	16	593
48	17	610
49	15	628
50	16	644
51	17	661
52	15	679
53	16	695
54	17	712
55	15	730
56	16	746
57	17	763
58	15	781
59	16	797
60	17	814
61	15	832
62	16	848
63	17	865
64	15	883
65	16	899
66	17	916
67	15	934
68	16	950
69	17	967
70	15	985
71	16	1001
72	17	1018
73	15	1036
74	16	1052
75	17	1069
76	15	1087
77	16	1103
78	17	1120
79	15	1138
80	16	1154
81	17	1171
82	15	1189
83	16	1205
84	17	1222
85	15	1240
86	16	1256
87	17	1273
88	15	1291
89	16	1307
90	17	1324
91	15	1342
92	16	1358
93	17	1375

[illegible]	[illegible]	[illegible]
94	15	1393
95	16	1409
96	17	1426
97	15	1444
98	16	1460
99	17	1477
100	15	1495
101	16	1511
102	17	1528
103	15	1546
104	16	1562
105	17	1579
106	15	1597
107	16	1613
108	17	1630
109	15	1648
110	16	1664
111	17	1681
112	15	1699
113	16	1715
114	17	1732
115	15	1750
116	16	1766
117	17	1783
118	15	1801
119	16	1817
120	17	1834
121	15	1852
122	16	1868
123	17	1885
124	15	1903
125	16	1919
126	17	1936
127	15	1954
128	16	1970
129	17	1987
130	15	2005
131	16	2021
132	17	2038
133	15	2056
134	16	2072
135	17	2089
136	15	2107
137	16	2123
138	17	2140
139	15	2158
140	16	2174
141	17	2191
142	15	2209
143	16	2225
144	17	2242
145	15	2260
146	16	2276
147	17	2293
148	15	2311
149	16	2327
150	17	2344
151	15	2362
152	16	2378
153	17	2395
154	15	2413
155	16	2429
156	17	2446

157	15	2464
158	16	2480
159	17	2497
160	15	2515
161	16	2531
162	17	2548
163	15	2566
164	16	2582
165	17	2599
166	15	2617
167	16	2633
168	17	2650
169	15	2668
170	16	2684
171	17	2701
172	15	2719
173	16	2735
174	17	2752
175	15	2770
176	16	2786
177	17	2803
178	15	2821
179	16	2837
180	17	2854
181	15	2872
182	16	2888
183	17	2905
184	15	2923
185	16	2939
186	17	2956
187	15	2974
188	16	2990
189	17	3007
190	15	3025
191	16	3041
192	17	3058
193	15	3076
194	16	3092
195	17	3109
196	15	3127
197	16	3143
198	17	3160
199	15	3178
200	16	3194
201	17	3211
202	15	3229
203	16	3245
204	17	3262
205	15	3280
206	16	3296
207	17	3313
208	15	3331
209	16	3347
210	17	3364
211	15	3382
212	16	3398
213	17	3415
214	15	3433
215	16	3449
216	17	3466
217	15	3484
218	16	3500
219	17	3517

```
       220                15                3535
       221                16                3551
       222                17                3568
       223                15                3586
       224                16                3602
       225                17                3619
       226                15                3637
       227                16                3653
       228                17                3670
       229                15                3688
       230                16                3704
       231                17                3721
       232                15                3739
       233                16                3755
       234                17                3772
       235                15                3790
       236                16                3806
       237                17                3823
       238                15                3841
       239                16                3857
       240                17                3874
       241                                  3892


     LSS =                                  3891


SS MATRIX
SS(   1)=    1260.416260  SS(    2)=    1260.416260  SS(    3)=        0.000000
SS(   4)=   99999.938000  SS(    5)=     624.999760  SS(    6)=      624.999760
SS(   7)=    2510.416000  SS(    8)=       0.000000  SS(    9)=        0.000000
SS(  10)=   -1249.999760  SS(   11)=    1270.833010  SS(   12)=        0.000000
SS(  13)=    -624.999760  SS(   14)=     -10.416666  SS(   15)=        0.000000
SS(  16)= 149999.875000   SS(   17)=       0.000000  SS(   18)=      624.999760
SS(  19)=  24999.992200   SS(   20)=     624.999760  SS(   21)=        0.000000
SS(  22)=    2510.416000  SS(   23)=       0.000000  SS(   24)=        0.000000
SS(  25)=   -1249.999760  SS(   26)=    1270.833010  SS(   27)=        0.000000
SS(  28)=    -624.999760  SS(   29)=     -10.416666  SS(   30)=        0.000000
SS(  31)= 149999.875000   SS(   32)=       0.000000  SS(   33)=      624.999760
SS(  34)=  24999.992200   SS(   35)=     624.999760  SS(   36)=        0.000000
SS(  37)=    2510.416000  SS(   38)=       0.000000  SS(   39)=        0.000000
SS(  40)=   -1249.999760  SS(   41)=    1270.833010  SS(   42)=        0.000000
SS(  43)=    -624.999760  SS(   44)=     -10.416666  SS(   45)=        0.000000
SS(  46)= 149999.875000   SS(   47)=       0.000000  SS(   48)=      624.999760
SS(  49)=  24999.992200   SS(   50)=     624.999760  SS(   51)=        0.000000
SS(  52)=    1260.416260  SS(   53)=       0.000000  SS(   54)=        0.000000
SS(  55)=   -1249.999760  SS(   56)=    1260.416260  SS(   57)=        0.000000
SS(  58)=    -624.999760  SS(   59)=     -10.416666  SS(   60)=        0.000000
SS(  61)=   99999.938000  SS(   62)=    -624.999760  SS(   63)=      624.999760
SS(  64)=  24999.992200   SS(   65)=     624.999760  SS(   66)=        0.000000
SS(  67)=    1270.832760  SS(   68)=       0.000000  SS(   69)=        0.000000
SS(  70)=       0.000000  SS(   71)=       0.000000  SS(   72)=        0.000000
SS(  73)=       0.000000  SS(   74)=       0.000000  SS(   75)=        0.000000
SS(  76)=       0.000000  SS(   77)=       0.000000  SS(   78)=        0.000000
SS(  79)=       0.000000  SS(   80)=    -624.999760  SS(   81)=        0.000000
SS(  82)=     -10.416666  SS(   83)=    2510.416000  SS(   84)=        0.000000
SS(  85)=       0.000000  SS(   86)=       0.000000  SS(   87)=        0.000000
SS(  88)=       0.000000  SS(   89)=       0.000000  SS(   90)=        0.000000
SS(  91)=       0.000000  SS(   92)=       0.000000  SS(   93)=        0.000000
SS(  94)=       0.000000  SS(   95)=       0.000000  SS(   96)=        0.000000
SS(  97)=       0.000000  SS(   98)=   -1249.999760  SS(   99)=        0.000000
SS( 100)= 149999.875000   SS(  101)=     624.999760  SS(  102)=        0.000000
SS( 103)=       0.000000  SS(  104)=       0.000000  SS(  105)=        0.000000
SS( 106)=       0.000000 SS( 107)=       0.000000 SS( 108)=       0.000000
SS( 109)=       0.000000 SS( 110)=       0.000000 SS( 111)=       0.000000
SS( 112)=       0.000000 SS( 113)=       0.000000 SS( 114)=       0.000000
SS( 115)=   24999.992200 SS( 116)=       0.000000 SS( 117)=     624.999760
SS( 118)=    2520.832500 SS( 119)=       0.000000 SS( 120)=       0.000000
SS( 121)=   -1249.999760 SS( 122)=       0.000000 SS( 123)=       0.000000
SS( 124)=       0.000000 SS( 125)=       0.000000 SS( 126)=       0.000000
SS( 127)=       0.000000 SS( 128)=       0.000000 SS( 129)=       0.000000
SS( 130)=       0.000000 SS( 131)=    -624.999760 SS( 132)=       0.000000
SS( 133)=     -10.416666 SS( 134)=    2520.832500 SS( 135)=       0.000000
SS( 136)=    -624.999760 SS( 137)=     -10.416666 SS( 138)=       0.000000
SS( 139)=       0.000000 SS( 140)=       0.000000 SS( 141)=       0.000000
SS( 142)=       0.000000 SS( 143)=       0.000000 SS( 144)=       0.000000
SS( 145)=       0.000000 SS( 146)=       0.000000 SS( 147)=       0.000000
SS( 148)=       0.000000 SS( 149)=   -1249.999760 SS( 150)=       0.000000
SS( 151)=  199999.812000 SS( 152)=       0.000000 SS( 153)=       0.000000
SS( 154)=   24999.992200 SS( 155)=     624.999760 SS( 156)=       0.000000
SS( 157)=       0.000000 SS( 158)=       0.000000 SS( 159)=       0.000000
SS( 160)=       0.000000 SS( 161)=       0.000000 SS( 162)=       0.000000
SS( 163)=       0.000000 SS( 164)=       0.000000 SS( 165)=       0.000000
SS( 166)=   24999.992200 SS( 167)=       0.000000 SS( 168)=     624.999760
SS( 169)=    2520.832500 SS( 170)=       0.000000 SS( 171)=       0.000000
SS( 172)=   -1249.999760 SS( 173)=       0.000000 SS( 174)=       0.000000
SS( 175)=       0.000000 SS( 176)=       0.000000 SS( 177)=       0.000000
SS( 178)=       0.000000 SS( 179)=       0.000000 SS( 180)=       0.000000
SS( 181)=       0.000000 SS( 182)=    -624.999760 SS( 183)=       0.000000
SS( 184)=     -10.416666 SS( 185)=    2520.832500 SS( 186)=       0.000000
SS( 187)=    -624.999760 SS( 188)=     -10.416666 SS( 189)=       0.000000
SS( 190)=       0.000000 SS( 191)=       0.000000 SS( 192)=       0.000000
SS( 193)=       0.000000 SS( 194)=       0.000000 SS( 195)=       0.000000
SS( 196)=       0.000000 SS( 197)=       0.000000 SS( 198)=       0.000000
SS( 199)=       0.000000 SS( 200)=   -1249.999760 SS( 201)=       0.000000
SS( 202)=  199999.812000 SS( 203)=       0.000000 SS( 204)=       0.000000
SS( 205)=   24999.992200 SS( 206)=     624.999760 SS( 207)=       0.000000
SS( 208)=       0.000000 SS( 209)=       0.000000 SS( 210)=       0.000000
SS( 211)=       0.000000 SS( 212)=       0.000000 SS( 213)=       0.000000
SS( 214)=       0.000000 SS( 215)=       0.000000 SS( 216)=       0.000000
SS( 217)=   24999.992200 SS( 218)=       0.000000 SS( 219)=     624.999760
SS( 220)=    2520.832500 SS( 221)=       0.000000 SS( 222)=       0.000000
SS( 223)=   -1249.999760 SS( 224)=       0.000000 SS( 225)=       0.000000
SS( 226)=       0.000000 SS( 227)=       0.000000 SS( 228)=       0.000000
SS( 229)=       0.000000 SS( 230)=       0.000000 SS( 231)=       0.000000
SS( 232)=       0.000000 SS( 233)=    -624.999760 SS( 234)=       0.000000
SS( 235)=     -10.416666 SS( 236)=    2520.832500 SS( 237)=       0.000000
SS( 238)=    -624.999760 SS( 239)=     -10.416666 SS( 240)=       0.000000
SS( 241)=       0.000000 SS( 242)=       0.000000 SS( 243)=       0.000000
SS( 244)=       0.000000 SS( 245)=       0.000000 SS( 246)=       0.000000
SS( 247)=       0.000000 SS( 248)=       0.000000 SS( 249)=       0.000000
SS( 250)=       0.000000 SS( 251)=   -1249.999760 SS( 252)=       0.000000
SS( 253)=  199999.812000 SS( 254)=       0.000000 SS( 255)=       0.000000
SS( 256)=   24999.992200 SS( 257)=     624.999760 SS( 258)=       0.000000
SS( 259)=       0.000000 SS( 260)=       0.000000 SS( 261)=       0.000000
SS( 262)=       0.000000 SS( 263)=       0.000000 SS( 264)=       0.000000
SS( 265)=       0.000000 SS( 266)=       0.000000 SS( 267)=       0.000000
SS( 268)=   24999.992200 SS( 269)=       0.000000 SS( 270)=     624.999760
SS( 271)=    1270.832760 SS( 272)=       0.000000 SS( 273)=       0.000000
SS( 274)=   -1249.999760 SS( 275)=       0.000000 SS( 276)=       0.000000
SS( 277)=       0.000000 SS( 278)=       0.000000 SS( 279)=       0.000000
SS( 280)=       0.000000 SS( 281)=       0.000000 SS( 282)=       0.000000
SS( 283)=       0.000000 SS( 284)=    -624.999760 SS( 285)=       0.000000
SS( 286)=     -10.416666 SS( 287)=    2510.416000 SS( 288)=       0.000000
SS( 289)=    -624.999760 SS( 290)=     -10.416666 SS( 291)=       0.000000
SS( 292)=       0.000000 SS( 293)=       0.000000 SS( 294)=       0.000000
SS( 295)=      0.000000  SS( 296)=      0.000000  SS( 297)=      0.000000
SS( 298)=      0.000000  SS( 299)=      0.000000  SS( 300)=      0.000000
SS( 301)=      0.000000  SS( 302)=  -1249.999760  SS( 303)=      0.000000
SS( 304)= 149999.875000  SS( 305)=   -624.999760  SS( 306)=      0.000000
SS( 307)=  24999.992200  SS( 308)=    624.999760  SS( 309)=      0.000000
SS( 310)=      0.000000  SS( 311)=      0.000000  SS( 312)=      0.000000
SS( 313)=      0.000000  SS( 314)=      0.000000  SS( 315)=      0.000000
SS( 316)=      0.000000  SS( 317)=      0.000000  SS( 318)=      0.000000
SS( 319)=  24999.992200  SS( 320)=      0.000000  SS( 321)=    624.999760
SS( 322)=   1270.832760  SS( 323)=      0.000000  SS( 324)=      0.000000
SS( 325)=      0.000000  SS( 326)=      0.000000  SS( 327)=      0.000000
SS( 328)=      0.000000  SS( 329)=      0.000000  SS( 330)=      0.000000
SS( 331)=      0.000000  SS( 332)=      0.000000  SS( 333)=      0.000000
SS( 334)=      0.000000  SS( 335)=   -624.999760  SS( 336)=      0.000000
SS( 337)=    -10.416666  SS( 338)=   2510.416000  SS( 339)=      0.000000
SS( 340)=      0.000000  SS( 341)=      0.000000  SS( 342)=      0.000000
SS( 343)=      0.000000  SS( 344)=      0.000000  SS( 345)=      0.000000
SS( 346)=      0.000000  SS( 347)=      0.000000  SS( 348)=      0.000000
SS( 349)=      0.000000  SS( 350)=      0.000000  SS( 351)=      0.000000
SS( 352)=      0.000000  SS( 353)=  -1249.999760  SS( 354)=      0.000000
SS( 355)= 149999.875000  SS( 356)=    624.999760  SS( 357)=      0.000000
SS( 358)=      0.000000  SS( 359)=      0.000000  SS( 360)=      0.000000
SS( 361)=      0.000000  SS( 362)=      0.000000  SS( 363)=      0.000000
SS( 364)=      0.000000  SS( 365)=      0.000000  SS( 366)=      0.000000
SS( 367)=      0.000000  SS( 368)=      0.000000  SS( 369)=      0.000000
SS( 370)=  24999.992200  SS( 371)=      0.000000  SS( 372)=    624.999760
SS( 373)=   2520.832500  SS( 374)=      0.000000  SS( 375)=      0.000000
SS( 376)=  -1249.999760  SS( 377)=      0.000000  SS( 378)=      0.000000
SS( 379)=      0.000000  SS( 380)=      0.000000  SS( 381)=      0.000000
SS( 382)=      0.000000  SS( 383)=      0.000000  SS( 384)=      0.000000
SS( 385)=      0.000000  SS( 386)=   -624.999760  SS( 387)=      0.000000
SS( 388)=    -10.416666  SS( 389)=   2520.832500  SS( 390)=      0.000000
SS( 391)=   -624.999760  SS( 392)=    -10.416666  SS( 393)=      0.000000
SS( 394)=      0.000000  SS( 395)=      0.000000  SS( 396)=      0.000000
SS( 397)=      0.000000  SS( 398)=      0.000000  SS( 399)=      0.000000
SS( 400)=      0.000000  SS( 401)=      0.000000  SS( 402)=      0.000000
SS( 403)=      0.000000  SS( 404)=  -1249.999760  SS( 405)=      0.000000
SS( 406)= 199999.812000  SS( 407)=      0.000000  SS( 408)=      0.000000
SS( 409)=  24999.992200  SS( 410)=    624.999760  SS( 411)=      0.000000
SS( 412)=      0.000000  SS( 413)=      0.000000  SS( 414)=      0.000000
SS( 415)=      0.000000  SS( 416)=      0.000000  SS( 417)=      0.000000
SS( 418)=      0.000000  SS( 419)=      0.000000  SS( 420)=      0.000000
SS( 421)=  24999.992200  SS( 422)=      0.000000  SS( 423)=    624.999760
SS( 424)=   2520.832500  SS( 425)=      0.000000  SS( 426)=      0.000000
SS( 427)=  -1249.999760  SS( 428)=      0.000000  SS( 429)=      0.000000
SS( 430)=      0.000000  SS( 431)=      0.000000  SS( 432)=      0.000000
SS( 433)=      0.000000  SS( 434)=      0.000000  SS( 435)=      0.000000
SS( 436)=      0.000000  SS( 437)=   -624.999760  SS( 438)=      0.000000
SS( 439)=    -10.416666  SS( 440)=   2520.832500  SS( 441)=      0.000000
SS( 442)=   -624.999760  SS( 443)=    -10.416666  SS( 444)=      0.000000
SS( 445)=      0.000000  SS( 446)=      0.000000  SS( 447)=      0.000000
SS( 448)=      0.000000  SS( 449)=      0.000000  SS( 450)=      0.000000
SS( 451)=      0.000000  SS( 452)=      0.000000  SS( 453)=      0.000000
SS( 454)=      0.000000  SS( 455)=  -1249.999760  SS( 456)=      0.000000
SS( 457)= 199999.812000  SS( 458)=      0.000000  SS( 459)=      0.000000
SS( 460)=  24999.992200  SS( 461)=    624.999760  SS( 462)=      0.000000
SS( 463)=      0.000000  SS( 464)=      0.000000  SS( 465)=      0.000000
SS( 466)=      0.000000  SS( 467)=      0.000000  SS( 468)=      0.000000
SS( 469)=      0.000000  SS( 470)=      0.000000  SS( 471)=      0.000000
SS( 472)=  24999.992200  SS( 473)=      0.000000  SS( 474)=    624.999760
SS( 475)=   2520.832500  SS( 476)=      0.000000  SS( 477)=      0.000000
SS( 478)=  -1249.999760  SS( 479)=      0.000000  SS( 480)=      0.000000
SS( 481)=      0.000000  SS( 482)=      0.000000  SS( 483)=      0.000000
SS( 484)=       0.000000   SS( 485)=       0.000000   SS( 486)=       0.000000
SS( 487)=       0.000000   SS( 488)=    -624.999760   SS( 489)=       0.000000
SS( 490)=     -10.416666   SS( 491)=    2520.832500   SS( 492)=       0.000000
SS( 493)=    -624.999760   SS( 494)=     -10.416666   SS( 495)=       0.000000
SS( 496)=       0.000000   SS( 497)=       0.000000   SS( 498)=       0.000000
SS( 499)=       0.000000   SS( 500)=       0.000000   SS( 501)=       0.000000
SS( 502)=       0.000000   SS( 503)=       0.000000   SS( 504)=       0.000000
SS( 505)=       0.000000   SS( 506)=   -1249.999760   SS( 507)=       0.000000
SS( 508)= 199999.812000   SS( 509)=       0.000000   SS( 510)=       0.000000
SS( 511)=  24999.992200   SS( 512)=     624.999760   SS( 513)=       0.000000
SS( 514)=       0.000000   SS( 515)=       0.000000   SS( 516)=       0.000000
SS( 517)=       0.000000   SS( 518)=       0.000000   SS( 519)=       0.000000
SS( 520)=       0.000000   SS( 521)=       0.000000   SS( 522)=       0.000000
SS( 523)=  24999.992200   SS( 524)=       0.000000   SS( 525)=     624.999760
SS( 526)=    1270.832760   SS( 527)=       0.000000   SS( 528)=       0.000000
SS( 529)=   -1249.999760   SS( 530)=       0.000000   SS( 531)=       0.000000
SS( 532)=       0.000000   SS( 533)=       0.000000   SS( 534)=       0.000000
SS( 535)=       0.000000   SS( 536)=       0.000000   SS( 537)=       0.000000
SS( 538)=       0.000000   SS( 539)=    -624.999760   SS( 540)=       0.000000
SS( 541)=     -10.416666   SS( 542)=    2510.416000   SS( 543)=       0.000000
SS( 544)=    -624.999760   SS( 545)=     -10.416666   SS( 546)=       0.000000
SS( 547)=       0.000000   SS( 548)=       0.000000   SS( 549)=       0.000000
SS( 550)=       0.000000   SS( 551)=       0.000000   SS( 552)=       0.000000
SS( 553)=       0.000000   SS( 554)=       0.000000   SS( 555)=       0.000000
SS( 556)=       0.000000   SS( 557)=   -1249.999760   SS( 558)=       0.000000
SS( 559)= 149999.875000   SS( 560)=    -624.999760   SS( 561)=       0.000000
SS( 562)=  24999.992200   SS( 563)=     624.999760   SS( 564)=       0.000000
SS( 565)=       0.000000   SS( 566)=       0.000000   SS( 567)=       0.000000
SS( 568)=       0.000000   SS( 569)=       0.000000   SS( 570)=       0.000000
SS( 571)=       0.000000   SS( 572)=       0.000000   SS( 573)=       0.000000
SS( 574)=  24999.992200   SS( 575)=       0.000000   SS( 576)=     624.999760
SS( 577)=    1270.832760   SS( 578)=       0.000000   SS( 579)=       0.000000
SS( 580)=       0.000000   SS( 581)=       0.000000   SS( 582)=       0.000000
SS( 583)=       0.000000   SS( 584)=       0.000000   SS( 585)=       0.000000
SS( 586)=       0.000000   SS( 587)=       0.000000   SS( 588)=       0.000000
SS( 589)=       0.000000   SS( 590)=    -624.999760   SS( 591)=       0.000000
SS( 592)=     -10.416666   SS( 593)=    2510.416000   SS( 594)=       0.000000
SS( 595)=       0.000000   SS( 596)=       0.000000   SS( 597)=       0.000000
SS( 598)=       0.000000   SS( 599)=       0.000000   SS( 600)=       0.000000
SS( 601)=       0.000000   SS( 602)=       0.000000   SS( 603)=       0.000000
SS( 604)=       0.000000   SS( 605)=       0.000000   SS( 606)=       0.000000
SS( 607)=       0.000000   SS( 608)=   -1249.999760   SS( 609)=       0.000000
SS( 610)= 149999.875000   SS( 611)=     624.999760   SS( 612)=       0.000000
SS( 613)=       0.000000   SS( 614)=       0.000000   SS( 615)=       0.000000
SS( 616)=       0.000000   SS( 617)=       0.000000   SS( 618)=       0.000000
SS( 619)=       0.000000   SS( 620)=       0.000000   SS( 621)=       0.000000
SS( 622)=       0.000000   SS( 623)=       0.000000   SS( 624)=       0.000000
SS( 625)=  24999.992200   SS( 626)=       0.000000   SS( 627)=     624.999760
SS( 628)=    2520.832500   SS( 629)=       0.000000   SS( 630)=       0.000000
SS( 631)=   -1249.999760   SS( 632)=       0.000000   SS( 633)=       0.000000
SS( 634)=       0.000000   SS( 635)=       0.000000   SS( 636)=       0.000000
SS( 637)=       0.000000   SS( 638)=       0.000000   SS( 639)=       0.000000
SS( 640)=       0.000000   SS( 641)=    -624.999760   SS( 642)=       0.000000
SS( 643)=     -10.416666   SS( 644)=    2520.832500   SS( 645)=       0.000000
SS( 646)=    -624.999760   SS( 647)=     -10.416666   SS( 648)=       0.000000
SS( 649)=       0.000000   SS( 650)=       0.000000   SS( 651)=       0.000000
SS( 652)=       0.000000   SS( 653)=       0.000000   SS( 654)=       0.000000
SS( 655)=       0.000000   SS( 656)=       0.000000   SS( 657)=       0.000000
SS( 658)=       0.000000   SS( 659)=   -1249.999760   SS( 660)=       0.000000
SS( 661)= 199999.812000   SS( 662)=       0.000000   SS( 663)=       0.000000
SS( 664)=  24999.992200   SS( 665)=     624.999760   SS( 666)=       0.000000
SS( 667)=       0.000000   SS( 668)=       0.000000   SS( 669)=       0.000000
SS( 670)=       0.000000   SS( 671)=       0.000000   SS( 672)=       0.000000
SS( 673)=      0.000000   SS( 674)=      0.000000   SS( 675)=      0.000000
SS( 676)=  24999.992200   SS( 677)=      0.000000   SS( 678)=    624.999760
SS( 679)=   2520.832500   SS( 680)=      0.000000   SS( 681)=      0.000000
SS( 682)=  -1249.999760   SS( 683)=      0.000000   SS( 684)=      0.000000
SS( 685)=      0.000000   SS( 686)=      0.000000   SS( 687)=      0.000000
SS( 688)=      0.000000   SS( 689)=      0.000000   SS( 690)=      0.000000
SS( 691)=      0.000000   SS( 692)=   -624.999760   SS( 693)=      0.000000
SS( 694)=    -10.416666   SS( 695)=   2520.832500   SS( 696)=      0.000000
SS( 697)=   -624.999760   SS( 698)=    -10.416666   SS( 699)=      0.000000
SS( 700)=      0.000000   SS( 701)=      0.000000   SS( 702)=      0.000000
SS( 703)=      0.000000   SS( 704)=      0.000000   SS( 705)=      0.000000
SS( 706)=      0.000000   SS( 707)=      0.000000   SS( 708)=      0.000000
SS( 709)=      0.000000   SS( 710)=  -1249.999760   SS( 711)=      0.000000
SS( 712)= 199999.812000   SS( 713)=      0.000000   SS( 714)=      0.000000
SS( 715)=  24999.992200   SS( 716)=    624.999760   SS( 717)=      0.000000
SS( 718)=      0.000000   SS( 719)=      0.000000   SS( 720)=      0.000000
SS( 721)=      0.000000   SS( 722)=      0.000000   SS( 723)=      0.000000
SS( 724)=      0.000000   SS( 725)=      0.000000   SS( 726)=      0.000000
SS( 727)=  24999.992200   SS( 728)=      0.000000   SS( 729)=    624.999760
SS( 730)=   2520.832500   SS( 731)=      0.000000   SS( 732)=      0.000000
SS( 733)=  -1249.999760   SS( 734)=      0.000000   SS( 735)=      0.000000
SS( 736)=      0.000000   SS( 737)=      0.000000   SS( 738)=      0.000000
SS( 739)=      0.000000   SS( 740)=      0.000000   SS( 741)=      0.000000
SS( 742)=      0.000000   SS( 743)=   -624.999760   SS( 744)=      0.000000
SS( 745)=    -10.416666   SS( 746)=   2520.832500   SS( 747)=      0.000000
SS( 748)=   -624.999760   SS( 749)=    -10.416666   SS( 750)=      0.000000
SS( 751)=      0.000000   SS( 752)=      0.000000   SS( 753)=      0.000000
SS( 754)=      0.000000   SS( 755)=      0.000000   SS( 756)=      0.000000
SS( 757)=      0.000000   SS( 758)=      0.000000   SS( 759)=      0.000000
SS( 760)=      0.000000   SS( 761)=  -1249.999760   SS( 762)=      0.000000
SS( 763)= 199999.812000   SS( 764)=      0.000000   SS( 765)=      0.000000
SS( 766)=  24999.992200   SS( 767)=    624.999760   SS( 768)=      0.000000
SS( 769)=      0.000000   SS( 770)=      0.000000   SS( 771)=      0.000000
SS( 772)=      0.000000   SS( 773)=      0.000000   SS( 774)=      0.000000
SS( 775)=      0.000000   SS( 776)=      0.000000   SS( 777)=      0.000000
SS( 778)=  24999.992200   SS( 779)=      0.000000   SS( 780)=    624.999760
SS( 781)=   1270.832760   SS( 782)=      0.000000   SS( 783)=      0.000000
SS( 784)=  -1249.999760   SS( 785)=      0.000000   SS( 786)=      0.000000
SS( 787)=      0.000000   SS( 788)=      0.000000   SS( 789)=      0.000000
SS( 790)=      0.000000   SS( 791)=      0.000000   SS( 792)=      0.000000
SS( 793)=      0.000000   SS( 794)=   -624.999760   SS( 795)=      0.000000
SS( 796)=    -10.416666   SS( 797)=   2510.416000   SS( 798)=      0.000000
SS( 799)=   -624.999760   SS( 800)=    -10.416666   SS( 801)=      0.000000
SS( 802)=      0.000000   SS( 803)=      0.000000   SS( 804)=      0.000000
SS( 805)=      0.000000   SS( 806)=      0.000000   SS( 807)=      0.000000
SS( 808)=      0.000000   SS( 809)=      0.000000   SS( 810)=      0.000000
SS( 811)=      0.000000   SS( 812)=  -1249.999760   SS( 813)=      0.000000
SS( 814)= 149999.875000   SS( 815)=   -624.999760   SS( 816)=      0.000000
SS( 817)=  24999.992200   SS( 818)=    624.999760   SS( 819)=      0.000000
SS( 820)=      0.000000   SS( 821)=      0.000000   SS( 822)=      0.000000
SS( 823)=      0.000000   SS( 824)=      0.000000   SS( 825)=      0.000000
SS( 826)=      0.000000   SS( 827)=      0.000000   SS( 828)=      0.000000
SS( 829)=  24999.992200   SS( 830)=      0.000000   SS( 831)=    624.999760
SS( 832)=   1270.832760   SS( 833)=      0.000000   SS( 834)=      0.000000
SS( 835)=      0.000000   SS( 836)=      0.000000   SS( 837)=      0.000000
SS( 838)=      0.000000   SS( 839)=      0.000000   SS( 840)=      0.000000
SS( 841)=      0.000000   SS( 842)=      0.000000   SS( 843)=      0.000000
SS( 844)=      0.000000   SS( 845)=   -624.999760   SS( 846)=      0.000000
SS( 847)=    -10.416666   SS( 848)=   2510.416000   SS( 849)=      0.000000
SS( 850)=      0.000000   SS( 851)=      0.000000   SS( 852)=      0.000000
SS( 853)=      0.000000   SS( 854)=      0.000000   SS( 855)=      0.000000
SS( 856)=      0.000000   SS( 857)=      0.000000   SS( 858)=      0.000000
SS( 859)=      0.000000   SS( 860)=      0.000000   SS( 861)=      0.000000
SS( 862)=      0.000000 SS( 863)=  -1249.999760 $S( 864)=       0.000000
SS( 865)= 149999.875000 SS( 866)=    624.999760 $S( 867)=       0.000000
SS( 868)=      0.000000 SS( 869)=      0.000000 SS( 870)=       0.000000
SS( 871)=      0.000000 SS( 872)=      0.000000 SS( 873)=       0.000000
SS( 874)=      0.000000 SS( 875)=      0.000000 SS( 876)=       0.000000
SS( 877)=      0.000000 SS( 878)=      0.000000 SS( 879)=       0.000000
SS( 880)=  24999.992200 SS( 881)=      0.000000 SS( 882)=     624.999760
SS( 883)=   2520.832500 SS( 884)=      0.000000 SS( 885)=       0.000000
SS( 886)=  -1249.999760 SS( 887)=      0.000000 SS( 888)=       0.000000
SS( 889)=      0.000000 SS( 890)=      0.000000 SS( 891)=       0.000000
SS( 892)=      0.000000 SS( 893)=      0.000000 SS( 894)=       0.000000
SS( 895)=      0.000000 SS( 896)=   -624.999760 SS( 897)=       0.000000
SS( 898)=    -10.416666 SS( 899)=   2520.832500 SS( 900)=       0.000000
SS( 901)=   -624.999760 SS( 902)=    -10.416666 SS( 903)=       0.000000
SS( 904)=      0.000000 SS( 905)=      0.000000 SS( 906)=       0.000000
SS( 907)=      0.000000 SS( 908)=      0.000000 SS( 909)=       0.000000
SS( 910)=      0.000000 SS( 911)=      0.000000 SS( 912)=       0.000000
SS( 913)=      0.000000 SS( 914)=  -1249.999760 SS( 915)=       0.000000
SS( 916)= 199999.812000 SS( 917)=      0.000000 SS( 918)=       0.000000
SS( 919)=  24999.992200 SS( 920)=    624.999760 SS( 921)=       0.000000
SS( 922)=      0.000000 SS( 923)=      0.000000 SS( 924)=       0.000000
SS( 925)=      0.000000 SS( 926)=      0.000000 SS( 927)=       0.000000
SS( 928)=      0.000000 SS( 929)=      0.000000 SS( 930)=       0.000000
SS( 931)=  24999.992200 SS( 932)=      0.000000 SS( 933)=     624.999760
SS( 934)=   2520.832500 SS( 935)=      0.000000 SS( 936)=       0.000000
SS( 937)=  -1249.999760 SS( 938)=      0.000000 SS( 939)=       0.000000
SS( 940)=      0.000000 SS( 941)=      0.000000 SS( 942)=       0.000000
SS( 943)=      0.000000 SS( 944)=      0.000000 SS( 945)=       0.000000
SS( 946)=      0.000000 SS( 947)=   -624.999760 SS( 948)=       0.000000
SS( 949)=    -10.416666 SS( 950)=   2520.832500 SS( 951)=       0.000000
SS( 952)=   -624.999760 SS( 953)=    -10.416666 SS( 954)=       0.000000
SS( 955)=      0.000000 SS( 956)=      0.000000 SS( 957)=       0.000000
SS( 958)=      0.000000 SS( 959)=      0.000000 SS( 960)=       0.000000
SS( 961)=      0.000000 SS( 962)=      0.000000 SS( 963)=       0.000000
SS( 964)=      0.000000 SS( 965)=  -1249.999760 SS( 966)=       0.000000
SS( 967)= 199999.812000 SS( 968)=      0.000000 SS( 969)=       0.000000
SS( 970)=  24999.992200 SS( 971)=    624.999760 SS( 972)=       0.000000
SS( 973)=      0.000000 SS( 974)=      0.000000 SS( 975)=       0.000000
SS( 976)=      0.000000 SS( 977)=      0.000000 SS( 978)=       0.000000
SS( 979)=      0.000000 SS( 980)=      0.000000 SS( 981)=       0.000000
SS( 982)=  24999.992200 SS( 983)=      0.000000 SS( 984)=     624.999760
SS( 985)=   2520.832500 SS( 986)=      0.000000 SS( 987)=       0.000000
SS( 988)=  -1249.999760 SS( 989)=      0.000000 SS( 990)=       0.000000
SS( 991)=      0.000000 SS( 992)=      0.000000 SS( 993)=       0.000000
SS( 994)=      0.000000 SS( 995)=      0.000000 SS( 996)=       0.000000
SS( 997)=      0.000000 SS( 998)=   -624.999760 SS( 999)=       0.000000
SS(1000)=    -10.416666 SS(1001)=   2520.832500 SS(1002)=       0.000000
SS(1003)=   -624.999760 SS(1004)=    -10.416666 SS(1005)=       0.000000
SS(1006)=      0.000000 SS(1007)=      0.000000 SS(1008)=       0.000000
SS(1009)=      0.000000 SS(1010)=      0.000000 SS(1011)=       0.000000
SS(1012)=      0.000000 SS(1013)=      0.000000 SS(1014)=       0.000000
SS(1015)=      0.000000 SS(1016)=  -1249.999760 SS(1017)=       0.000000
SS(1018)= 199999.812000 SS(1019)=      0.000000 SS(1020)=       0.000000
SS(1021)=  24999.992200 SS(1022)=    624.999760 SS(1023)=       0.000000
SS(1024)=      0.000000 SS(1025)=      0.000000 SS(1026)=       0.000000
SS(1027)=      0.000000 SS(1028)=      0.000000 SS(1029)=       0.000000
SS(1030)=      0.000000 SS(1031)=      0.000000 SS(1032)=       0.000000
SS(1033)=  24999.992200 SS(1034)=      0.000000 SS(1035)=     624.999760
SS(1036)=   1270.832760 SS(1037)=      0.000000 SS(1038)=       0.000000
SS(1039)=  -1249.999760 SS(1040)=      0.000000 SS(1041)=       0.000000
SS(1042)=      0.000000 SS(1043)=      0.000000 SS(1044)=       0.000000
SS(1045)=      0.000000 SS(1046)=      0.000000 SS(1047)=       0.000000
SS(1048)=      0.000000 SS(1049)=   -624.999760 SS(1050)=       0.000000
SS(1051)=     -10.416666  SS(1052)=     2510.416000  SS(1053)=        0.000000
SS(1054)=    -624.999760  SS(1055)=      -10.416666  SS(1056)=        0.000000
SS(1057)=       0.000000  SS(1058)=        0.000000  SS(1059)=        0.000000
SS(1060)=       0.000000  SS(1061)=        0.000000  SS(1062)=        0.000000
SS(1063)=       0.000000  SS(1064)=        0.000000  SS(1065)=        0.000000
SS(1066)=       0.000000  SS(1067)=    -1249.999760  SS(1068)=        0.000000
SS(1069)= 149999.875000   SS(1070)=     -624.999760  SS(1071)=        0.000000
SS(1072)=  24999.992200   SS(1073)=      624.999760  SS(1074)=        0.000000
SS(1075)=       0.000000  SS(1076)=        0.000000  SS(1077)=        0.000000
SS(1078)=       0.000000  SS(1079)=        0.000000  SS(1080)=        0.000000
SS(1081)=       0.000000  SS(1082)=        0.000000  SS(1083)=        0.000000
SS(1084)=  24999.992200   SS(1085)=        0.000000  SS(1086)=      624.999760
SS(1087)=    1270.832760  SS(1088)=        0.000000  SS(1089)=        0.000000
SS(1090)=       0.000000  SS(1091)=        0.000000  SS(1092)=        0.000000
SS(1093)=       0.000000  SS(1094)=        0.000000  SS(1095)=        0.000000
SS(1096)=       0.000000  SS(1097)=        0.000000  SS(1098)=        0.000000
SS(1099)=       0.000000  SS(1100)=     -624.999760  SS(1101)=        0.000000
SS(1102)=     -10.416666  SS(1103)=     2510.416000  SS(1104)=        0.000000
SS(1105)=       0.000000  SS(1106)=        0.000000  SS(1107)=        0.000000
SS(1108)=       0.000000  SS(1109)=        0.000000  SS(1110)=        0.000000
SS(1111)=       0.000000  SS(1112)=        0.000000  SS(1113)=        0.000000
SS(1114)=       0.000000  SS(1115)=        0.000000  SS(1116)=        0.000000
SS(1117)=       0.000000  SS(1118)=    -1249.999760  SS(1119)=        0.000000
SS(1120)= 149999.875000   SS(1121)=      624.999760  SS(1122)=        0.000000
SS(1123)=       0.000000  SS(1124)=        0.000000  SS(1125)=        0.000000
SS(1126)=       0.000000  SS(1127)=        0.000000  SS(1128)=        0.000000
SS(1129)=       0.000000  SS(1130)=        0.000000  SS(1131)=        0.000000
SS(1132)=       0.000000  SS(1133)=        0.000000  SS(1134)=        0.000000
SS(1135)=  24999.992200   SS(1136)=        0.000000  SS(1137)=      624.999760
SS(1138)=    2520.832500  SS(1139)=        0.000000  SS(1140)=        0.000000
SS(1141)=   -1249.999760  SS(1142)=        0.000000  SS(1143)=        0.000000
SS(1144)=       0.000000  SS(1145)=        0.000000  SS(1146)=        0.000000
SS(1147)=       0.000000  SS(1148)=        0.000000  SS(1149)=        0.000000
SS(1150)=       0.000000  SS(1151)=     -624.999760  SS(1152)=        0.000000
SS(1153)=     -10.416666  SS(1154)=     2520.832500  SS(1155)=        0.000000
SS(1156)=    -624.999760  SS(1157)=      -10.416666  SS(1158)=        0.000000
SS(1159)=       0.000000  SS(1160)=        0.000000  SS(1161)=        0.000000
SS(1162)=       0.000000  SS(1163)=        0.000000  SS(1164)=        0.000000
SS(1165)=       0.000000  SS(1166)=        0.000000  SS(1167)=        0.000000
SS(1168)=       0.000000  SS(1169)=    -1249.999760  SS(1170)=        0.000000
SS(1171)= 199999.812000   SS(1172)=        0.000000  SS(1173)=        0.000000
SS(1174)=  24999.992200   SS(1175)=      624.999760  SS(1176)=        0.000000
SS(1177)=       0.000000  SS(1178)=        0.000000  SS(1179)=        0.000000
SS(1180)=       0.000000  SS(1181)=        0.000000  SS(1182)=        0.000000
SS(1183)=       0.000000  SS(1184)=        0.000000  SS(1185)=        0.000000
SS(1186)=  24999.992200   SS(1187)=        0.000000  SS(1188)=      624.999760
SS(1189)=    2520.832500  SS(1190)=        0.000000  SS(1191)=        0.000000
SS(1192)=   -1249.999760  SS(1193)=        0.000000  SS(1194)=        0.000000
SS(1195)=       0.000000  SS(1196)=        0.000000  SS(1197)=        0.000000
SS(1198)=       0.000000  SS(1199)=        0.000000  SS(1200)=        0.000000
SS(1201)=       0.000000  SS(1202)=     -624.999760  SS(1203)=        0.000000
SS(1204)=     -10.416666  SS(1205)=     2520.832500  SS(1206)=        0.000000
SS(1207)=    -624.999760  SS(1208)=      -10.416666  SS(1209)=        0.000000
SS(1210)=       0.000000  SS(1211)=        0.000000  SS(1212)=        0.000000
SS(1213)=       0.000000  SS(1214)=        0.000000  SS(1215)=        0.000000
SS(1216)=       0.000000  SS(1217)=        0.000000  SS(1218)=        0.000000
SS(1219)=       0.000000  SS(1220)=    -1249.999760  SS(1221)=        0.000000
SS(1222)= 199999.812000   SS(1223)=        0.000000  SS(1224)=        0.000000
SS(1225)=  24999.992200   SS(1226)=      624.999760  SS(1227)=        0.000000
SS(1228)=       0.000000  SS(1229)=        0.000000  SS(1230)=        0.000000
SS(1231)=       0.000000  SS(1232)=        0.000000  SS(1233)=        0.000000
SS(1234)=       0.000000  SS(1235)=        0.000000  SS(1236)=        0.000000
SS(1237)=  24999.992200   SS(1238)=        0.000000  SS(1239)=      624.999760
SS(1240)=  2520.832500  SS(1241)=      0.000000  SS(1242)=      0.000000
SS(1243)= -1249.999760  SS(1244)=      0.000000  SS(1245)=      0.000000
SS(1246)=     0.000000  SS(1247)=      0.000000  SS(1248)=      0.000000
SS(1249)=     0.000000  SS(1250)=      0.000000  SS(1251)=      0.000000
SS(1252)=     0.000000  SS(1253)=   -624.999760  SS(1254)=      0.000000
SS(1255)=   -10.416666  SS(1256)=   2520.832500  SS(1257)=      0.000000
SS(1258)=  -624.999760  SS(1259)=    -10.416666  SS(1260)=      0.000000
SS(1261)=     0.000000  SS(1262)=      0.000000  SS(1263)=      0.000000
SS(1264)=     0.000000  SS(1265)=      0.000000  SS(1266)=      0.000000
SS(1267)=     0.000000  SS(1268)=      0.000000  SS(1269)=      0.000000
SS(1270)=     0.000000  SS(1271)=  -1249.999760  SS(1272)=      0.000000
SS(1273)= 199999.812000 SS(1274)=      0.000000  SS(1275)=      0.000000
SS(1276)=  24999.992200 SS(1277)=    624.999760  SS(1278)=      0.000000
SS(1279)=     0.000000  SS(1280)=      0.000000  SS(1281)=      0.000000
SS(1282)=     0.000000  SS(1283)=      0.000000  SS(1284)=      0.000000
SS(1285)=     0.000000  SS(1286)=      0.000000  SS(1287)=      0.000000
SS(1288)=  24999.992200 SS(1289)=      0.000000  SS(1290)=    624.999760
SS(1291)=  1270.832760  SS(1292)=      0.000000  SS(1293)=      0.000000
SS(1294)= -1249.999760  SS(1295)=      0.000000  SS(1296)=      0.000000
SS(1297)=     0.000000  SS(1298)=      0.000000  SS(1299)=      0.000000
SS(1300)=     0.000000  SS(1301)=      0.000000  SS(1302)=      0.000000
SS(1303)=     0.000000  SS(1304)=   -624.999760  SS(1305)=      0.000000
SS(1306)=   -10.416666  SS(1307)=   2510.416000  SS(1308)=      0.000000
SS(1309)=  -624.999760  SS(1310)=    -10.416666  SS(1311)=      0.000000
SS(1312)=     0.000000  SS(1313)=      0.000000  SS(1314)=      0.000000
SS(1315)=     0.000000  SS(1316)=      0.000000  SS(1317)=      0.000000
SS(1318)=     0.000000  SS(1319)=      0.000000  SS(1320)=      0.000000
SS(1321)=     0.000000  SS(1322)=  -1249.999760  SS(1323)=      0.000000
SS(1324)= 149999.875000 SS(1325)=   -624.999760  SS(1326)=      0.000000
SS(1327)=  24999.992200 SS(1328)=    624.999760  SS(1329)=      0.000000
SS(1330)=     0.000000  SS(1331)=      0.000000  SS(1332)=      0.000000
SS(1333)=     0.000000  SS(1334)=      0.000000  SS(1335)=      0.000000
SS(1336)=     0.000000  SS(1337)=      0.000000  SS(1338)=      0.000000
SS(1339)=  24999.992200 SS(1340)=      0.000000  SS(1341)=    624.999760
SS(1342)=  1270.832760  SS(1343)=      0.000000  SS(1344)=      0.000000
SS(1345)=     0.000000  SS(1346)=      0.000000  SS(1347)=      0.000000
SS(1348)=     0.000000  SS(1349)=      0.000000  SS(1350)=      0.000000
SS(1351)=     0.000000  SS(1352)=      0.000000  SS(1353)=      0.000000
SS(1354)=     0.000000  SS(1355)=   -624.999760  SS(1356)=      0.000000
SS(1357)=   -10.416666  SS(1358)=   2510.416000  SS(1359)=      0.000000
SS(1360)=     0.000000  SS(1361)=      0.000000  SS(1362)=      0.000000
SS(1363)=     0.000000  SS(1364)=      0.000000  SS(1365)=      0.000000
SS(1366)=     0.000000  SS(1367)=      0.000000  SS(1368)=      0.000000
SS(1369)=     0.000000  SS(1370)=      0.000000  SS(1371)=      0.000000
SS(1372)=     0.000000  SS(1373)=  -1249.999760  SS(1374)=      0.000000
SS(1375)= 149999.875000 SS(1376)=    624.999760  SS(1377)=      0.000000
SS(1378)=     0.000000  SS(1379)=      0.000000  SS(1380)=      0.000000
SS(1381)=     0.000000  SS(1382)=      0.000000  SS(1383)=      0.000000
SS(1384)=     0.000000  SS(1385)=      0.000000  SS(1386)=      0.000000
SS(1387)=     0.000000  SS(1388)=      0.000000  SS(1389)=      0.000000
SS(1390)=  24999.992200 SS(1391)=      0.000000  SS(1392)=    624.999760
SS(1393)=  2520.832500  SS(1394)=      0.000000  SS(1395)=      0.000000
SS(1396)= -1249.999760  SS(1397)=      0.000000  SS(1398)=      0.000000
SS(1399)=     0.000000  SS(1400)=      0.000000  SS(1401)=      0.000000
SS(1402)=     0.000000  SS(1403)=      0.000000  SS(1404)=      0.000000
SS(1405)=     0.000000  SS(1406)=   -624.999760  SS(1407)=      0.000000
SS(1408)=   -10.416666  SS(1409)=   2520.832500  SS(1410)=      0.000000
SS(1411)=  -624.999760  SS(1412)=    -10.416666  SS(1413)=      0.000000
SS(1414)=     0.000000  SS(1415)=      0.000000  SS(1416)=      0.000000
SS(1417)=     0.000000  SS(1418)=      0.000000  SS(1419)=      0.000000
SS(1420)=     0.000000  SS(1421)=      0.000000  SS(1422)=      0.000000
SS(1423)=     0.000000  SS(1424)=  -1249.999760  SS(1425)=      0.000000
SS(1426)= 199999.812000 SS(1427)=      0.000000  SS(1428)=      0.000000
SS(1429)=  24999.992200  SS(1430)=     624.999760  SS(1431)=        0.000000
SS(1432)=      0.000000  SS(1433)=       0.000000  SS(1434)=        0.000000
SS(1435)=      0.000000  SS(1436)=       0.000000  SS(1437)=        0.000000
SS(1438)=      0.000000  SS(1439)=       0.000000  SS(1440)=        0.000000
SS(1441)=  24999.992200  SS(1442)=       0.000000  SS(1443)=      624.999760
SS(1444)=   2520.832500  SS(1445)=       0.000000  SS(1446)=        0.000000
SS(1447)=  -1249.999760  SS(1448)=       0.000000  SS(1449)=        0.000000
SS(1450)=      0.000000  SS(1451)=       0.000000  SS(1452)=        0.000000
SS(1453)=      0.000000  SS(1454)=       0.000000  SS(1455)=        0.000000
SS(1456)=      0.000000  SS(1457)=    -624.999760  SS(1458)=        0.000000
SS(1459)=    -10.416666  SS(1460)=    2520.832500  SS(1461)=        0.000000
SS(1462)=   -624.999760  SS(1463)=     -10.416666  SS(1464)=        0.000000
SS(1465)=      0.000000  SS(1466)=       0.000000  SS(1467)=        0.000000
SS(1468)=      0.000000  SS(1469)=       0.000000  SS(1470)=        0.000000
SS(1471)=      0.000000  SS(1472)=       0.000000  SS(1473)=        0.000000
SS(1474)=      0.000000  SS(1475)=   -1249.999760  SS(1476)=        0.000000
SS(1477)= 199999.812000  SS(1478)=       0.000000  SS(1479)=        0.000000
SS(1480)=  24999.992200  SS(1481)=     624.999760  SS(1482)=        0.000000
SS(1483)=      0.000000  SS(1484)=       0.000000  SS(1485)=        0.000000
SS(1486)=      0.000000  SS(1487)=       0.000000  SS(1488)=        0.000000
SS(1489)=      0.000000  SS(1490)=       0.000000  SS(1491)=        0.000000
SS(1492)=  24999.992200  SS(1493)=       0.000000  SS(1494)=      624.999760
SS(1495)=   2520.832500  SS(1496)=       0.000000  SS(1497)=        0.000000
SS(1498)=  -1249.999760  SS(1499)=       0.000000  SS(1500)=        0.000000
SS(1501)=      0.000000  SS(1502)=       0.000000  SS(1503)=        0.000000
SS(1504)=      0.000000  SS(1505)=       0.000000  SS(1506)=        0.000000
SS(1507)=      0.000000  SS(1508)=    -624.999760  SS(1509)=        0.000000
SS(1510)=    -10.416666  SS(1511)=    2520.832500  SS(1512)=        0.000000
SS(1513)=   -624.999760  SS(1514)=     -10.416666  SS(1515)=        0.000000
SS(1516)=      0.000000  SS(1517)=       0.000000  SS(1518)=        0.000000
SS(1519)=      0.000000  SS(1520)=       0.000000  SS(1521)=        0.000000
SS(1522)=      0.000000  SS(1523)=       0.000000  SS(1524)=        0.000000
SS(1525)=      0.000000  SS(1526)=   -1249.999760  SS(1527)=        0.000000
SS(1528)= 199999.812000  SS(1529)=       0.000000  SS(1530)=        0.000000
SS(1531)=  24999.992200  SS(1532)=     624.999760  SS(1533)=        0.000000
SS(1534)=      0.000000  SS(1535)=       0.000000  SS(1536)=        0.000000
SS(1537)=      0.000000  SS(1538)=       0.000000  SS(1539)=        0.000000
SS(1540)=      0.000000  SS(1541)=       0.000000  SS(1542)=        0.000000
SS(1543)=  24999.992200  SS(1544)=       0.000000  SS(1545)=      624.999760
SS(1546)=   1270.832760  SS(1547)=       0.000000  SS(1548)=        0.000000
SS(1549)=  -1249.999760  SS(1550)=       0.000000  SS(1551)=        0.000000
SS(1552)=      0.000000  SS(1553)=       0.000000  SS(1554)=        0.000000
SS(1555)=      0.000000  SS(1556)=       0.000000  SS(1557)=        0.000000
SS(1558)=      0.000000  SS(1559)=    -624.999760  SS(1560)=        0.000000
SS(1561)=    -10.416666  SS(1562)=    2510.416000  SS(1563)=        0.000000
SS(1564)=   -624.999760  SS(1565)=     -10.416666  SS(1566)=        0.000000
SS(1567)=      0.000000  SS(1568)=       0.000000  SS(1569)=        0.000000
SS(1570)=      0.000000  SS(1571)=       0.000000  SS(1572)=        0.000000
SS(1573)=      0.000000  SS(1574)=       0.000000  SS(1575)=        0.000000
SS(1576)=      0.000000  SS(1577)=   -1249.999760  SS(1578)=        0.000000
SS(1579)= 149999.875000  SS(1580)=    -624.999760  SS(1581)=        0.000000
SS(1582)=  24999.992200  SS(1583)=     624.999760  SS(1584)=        0.000000
SS(1585)=      0.000000  SS(1586)=       0.000000  SS(1587)=        0.000000
SS(1588)=      0.000000  SS(1589)=       0.000000  SS(1590)=        0.000000
SS(1591)=      0.000000  SS(1592)=       0.000000  SS(1593)=        0.000000
SS(1594)=  24999.992200  SS(1595)=       0.000000  SS(1596)=      624.999760
SS(1597)=   1270.832760  SS(1598)=       0.000000  SS(1599)=        0.000000
SS(1600)=      0.000000  SS(1601)=       0.000000  SS(1602)=        0.000000
SS(1603)=      0.000000  SS(1604)=       0.000000  SS(1605)=        0.000000
SS(1606)=      0.000000  SS(1607)=       0.000000  SS(1608)=        0.000000
SS(1609)=      0.000000  SS(1610)=    -624.999760  SS(1611)=        0.000000
SS(1612)=    -10.416666  SS(1613)=    2510.416000  SS(1614)=        0.000000
SS(1615)=      0.000000  SS(1616)=       0.000000  SS(1617)=        0.000000
SS(1618)=      0.000000 SS(1619)=      0.000000 SS(1620)=      0.000000
SS(1621)=      0.000000 SS(1622)=      0.000000 SS(1623)=      0.000000
SS(1624)=      0.000000 SS(1625)=      0.000000 SS(1626)=      0.000000
SS(1627)=      0.000000 SS(1628)=  -1249.999760 SS(1629)=      0.000000
SS(1630)= 149999.875000 SS(1631)=    624.999760 SS(1632)=      0.000000
SS(1633)=      0.000000 SS(1634)=      0.000000 SS(1635)=      0.000000
SS(1636)=      0.000000 SS(1637)=      0.000000 SS(1638)=      0.000000
SS(1639)=      0.000000 SS(1640)=      0.000000 SS(1641)=      0.000000
SS(1642)=      0.000000 SS(1643)=      0.000000 SS(1644)=      0.000000
SS(1645)=  24999.992200 SS(1646)=      0.000000 SS(1647)=    624.999760
SS(1648)=   2520.832500 SS(1649)=      0.000000 SS(1650)=      0.000000
SS(1651)=  -1249.999760 SS(1652)=      0.000000 SS(1653)=      0.000000
SS(1654)=      0.000000 SS(1655)=      0.000000 SS(1656)=      0.000000
SS(1657)=      0.000000 SS(1658)=      0.000000 SS(1659)=      0.000000
SS(1660)=      0.000000 SS(1661)=   -624.999760 SS(1662)=      0.000000
SS(1663)=    -10.416666 SS(1664)=   2520.832500 SS(1665)=      0.000000
SS(1666)=   -624.999760 SS(1667)=    -10.416666 SS(1668)=      0.000000
SS(1669)=      0.000000 SS(1670)=      0.000000 SS(1671)=      0.000000
SS(1672)=      0.000000 SS(1673)=      0.000000 SS(1674)=      0.000000
SS(1675)=      0.000000 SS(1676)=      0.000000 SS(1677)=      0.000000
SS(1678)=      0.000000 SS(1679)=  -1249.999760 SS(1680)=      0.000000
SS(1681)= 199999.812000 SS(1682)=      0.000000 SS(1683)=      0.000000
SS(1684)=  24999.992200 SS(1685)=    624.999760 SS(1686)=      0.000000
SS(1687)=      0.000000 SS(1688)=      0.000000 SS(1689)=      0.000000
SS(1690)=      0.000000 SS(1691)=      0.000000 SS(1692)=      0.000000
SS(1693)=      0.000000 SS(1694)=      0.000000 SS(1695)=      0.000000
SS(1696)=  24999.992200 SS(1697)=      0.000000 SS(1698)=    624.999760
SS(1699)=   2520.832500 SS(1700)=      0.000000 SS(1701)=      0.000000
SS(1702)=  -1249.999760 SS(1703)=      0.000000 SS(1704)=      0.0000
SS(1705)=      0.000000 SS(1706)=      0.000000 SS(1707)=      0.000
SS(1708)=      0.000000 SS(1709)=      0.000000 S:       =      0.000000
SS(1711)=      0.000000 SS(1712)=   -624.9           (1713)=      0.000000
SS(1714)=    -10.416666 SS(171:       520.832500 SS(1716)=      0.000000
SS(1717)=   -624         SS(1718)=    -10.416666 SS(1719)=      0.000000
SS(1720)=          )0000 SS(1721)=      0.000000 SS(1722)=      0.000000
SS(1723)=      0.000000 SS(1724)=      0.000000 SS(1725)=      0.000000
SS(1726)=      0.000000 SS(1727)=      0.000000 SS(1728)=      0.000000
SS(1729)=      0.000000 SS(1730)=  -1249.999760 SS(1731)=      0.000000
SS(1732)= 199999.812000 SS(1733)=      0.000000 SS(1734)=      0.000000
SS(1735)=  24999.992200 SS(1736)=    624.999760 SS(1737)=      0.000000
SS(1738)=      0.000000 SS(1739)=      0.000000 SS(1740)=      0.000000
SS(1741)=      0.000000 SS(1742)=      0.000000 SS(1743)=      0.000000
SS(1744)=      0.000000 SS(1745)=      0.000000 SS(1746)=      0.000000
SS(1747)=  24999.992200 SS(1748)=      0.000000 SS(1749)=    624.999760
SS(1750)=   2520.832500 SS(1751)=      0.000000 SS(1752)=      0.000000
SS(1753)=  -1249.999760 SS(1754)=      0.000000 SS(1755)=      0.000000
SS(1756)=      0.000000 SS(1757)=      0.000000 SS(1758)=      0.000000
SS(1759)=      0.000000 SS(1760)=      0.000000 SS(1761)=      0.000000
SS(1762)=      0.000000 SS(1763)=   -624.999760 SS(1764)=      0.000000
SS(1765)=    -10.416666 SS(1766)=   2520.832500 SS(1767)=      0.000000
SS(1768)=   -624.999760 SS(1769)=    -10.416666 SS(1770)=      0.000000
SS(1771)=      0.000000 SS(1772)=      0.000000 SS(1773)=      0.000000
SS(1774)=      0.000000 SS(1775)=      0.000000 SS(1776)=      0.000000
SS(1777)=      0.000000 SS(1778)=      0.000000 SS(1779)=      0.000000
SS(1780)=      0.000000 SS(1781)=  -1249.999760 SS(1782)=      0.000000
SS(1783)= 199999.812000 SS(1784)=      0.000000 SS(1785)=      0.000000
SS(1786)=  24999.992200 SS(1787)=    624.999760 SS(1788)=      0.000000
SS(1789)=      0.000000 SS(1790)=      0.000000 SS(1791)=      0.000000
SS(1792)=      0.000000 SS(1793)=      0.000000 SS(1794)=      0.000000
SS(1795)=      0.000000 SS(1796)=      0.000000 SS(1797)=      0.000000
SS(1798)=  24999.992200 SS(1799)=      0.000000 SS(1800)=    624.999760
SS(1801)=   1270.832760 SS(1802)=      0.000000 SS(1803)=      0.000000
SS(1804)=  -1249.999760 SS(1805)=      0.000000 SS(1806)=      0.000000
SS(1807)=       0.000000  SS(1808)=        0.000000  SS(1809)=        0.000000
SS(1810)=       0.000000  SS(1811)=        0.000000  SS(1812)=        0.000000
SS(1813)=       0.000000  SS(1814)=     -624.999760  SS(1815)=        0.000000
SS(1816)=     -10.416666  SS(1817)=     2510.416000  SS(1818)=        0.000000
SS(1819)=    -624.999760  SS(1820)=      -10.416666  SS(1821)=        0.000000
SS(1822)=       0.000000  SS(1823)=        0.000000  SS(1824)=        0.000000
SS(1825)=       0.000000  SS(1826)=        0.000000  SS(1827)=        0.000000
SS(1828)=       0.000000  SS(1829)=        0.000000  SS(1830)=        0.000000
SS(1831)=       0.000000  SS(1832)=    -1249.999760  SS(1833)=        0.000000
SS(1834)= 149999.875000  SS(1835)=     -624.999760  SS(1836)=        0.000000
SS(1837)=  24999.992200  SS(1838)=      624.999760  SS(1839)=        0.000000
SS(1840)=       0.000000  SS(1841)=        0.000000  SS(1842)=        0.000000
SS(1843)=       0.000000  SS(1844)=        0.000000  SS(1845)=        0.000000
SS(1846)=       0.000000  SS(1847)=        0.000000  SS(1848)=        0.000000
SS(1849)=  24999.992200  SS(1850)=        0.000000  SS(1851)=      624.999760
SS(1852)=    1270.832760  SS(1853)=        0.000000  SS(1854)=        0.000000
SS(1855)=       0.000000  SS(1856)=        0.000000  SS(1857)=        0.000000
SS(1858)=       0.000000  SS(1859)=        0.000000  SS(1860)=        0.000000
SS(1861)=       0.000000  SS(1862)=        0.000000  SS(1863)=        0.000000
SS(1864)=       0.000000  SS(1865)=     -624.999760  SS(1866)=        0.000000
SS(1867)=     -10.416666  SS(1868)=     2510.416000  SS(1869)=        0.000000
SS(1870)=       0.000000  SS(1871)=        0.000000  SS(1872)=        0.000000
SS(1873)=       0.000000  SS(1874)=        0.000000  SS(1875)=        0.000000
SS(1876)=       0.000000  SS(1877)=        0.000000  SS(1878)=        0.000000
SS(1879)=       0.000000  SS(1880)=        0.000000  SS(1881)=        0.000000
SS(1882)=       0.000000  SS(1883)=    -1249.999760  SS(1884)=        0.000000
SS(1885)= 149999.875000  SS(1886)=      624.999760  SS(1887)=        0.000000
SS(1888)=       0.000000  SS(1889)=        0.000000  SS(1890)=        0.000000
SS(1891)=       0.000000  SS(1892)=        0.000000  SS(1893)=        0.000000
SS(1894)=       0.000000  SS(1895)=        0.000000  SS(1896)=        0.000000
SS(1897)=       0.000000  SS(1898)=        0.000000  SS(1899)=        0.000000
SS(1900)=  24999.992200  SS(1901)=        0.000000  SS(1902)=      624.999760
SS(1903)=    2520.832500  SS(1904)=        0.000000  SS(1905)=        0.000000
SS(1906)=   -1249.999760  SS(1907)=        0.000000  SS(1908)=        0.000000
SS(1909)=       0.000000  SS(1910)=        0.000000  SS(1911)=        0.000000
SS(1912)=       0.000000  SS(1913)=        0.000000  SS(1914)=        0.000000
SS(1915)=       0.000000  SS(1916)=     -624.999760  SS(1917)=        0.000000
SS(1918)=     -10.416666  SS(1919)=     2520.832500  SS(1920)=        0.000000
SS(1921)=    -624.999760  SS(1922)=      -10.416666  SS(1923)=        0.000000
SS(1924)=       0.000000  SS(1925)=        0.000000  SS(1926)=        0.000000
SS(1927)=       0.000000  SS(1928)=        0.000000  SS(1929)=        0.000000
SS(1930)=       0.000000  SS(1931)=        0.000000  SS(1932)=        0.000000
SS(1933)=       0.000000  SS(1934)=    -1249.999760  SS(1935)=        0.000000
SS(1936)= 199999.812000  SS(1937)=        0.000000  SS(1938)=        0.000000
SS(1939)=  24999.992200  SS(1940)=      624.999760  SS(1941)=        0.000000
SS(1942)=       0.000000  SS(1943)=        0.000000  SS(1944)=        0.000000
SS(1945)=       0.000000  SS(1946)=        0.000000  SS(1947)=        0.000000
SS(1948)=       0.000000  SS(1949)=        0.000000  SS(1950)=        0.000000
SS(1951)=  24999.992200  SS(1952)=        0.000000  SS(1953)=      624.999760
SS(1954)=    2520.832500  SS(1955)=        0.000000  SS(1956)=        0.000000
SS(1957)=   -1249.999760  SS(1958)=        0.000000  SS(1959)=        0.000000
SS(1960)=       0.000000  SS(1961)=        0.000000  SS(1962)=        0.000000
SS(1963)=       0.000000  SS(1964)=        0.000000  SS(1965)=        0.000000
SS(1966)=       0.000000  SS(1967)=     -624.999760  SS(1968)=        0.000000
SS(1969)=     -10.416666  SS(1970)=     2520.832500  SS(1971)=        0.000000
SS(1972)=    -624.999760  SS(1973)=      -10.416666  SS(1974)=        0.000000
SS(1975)=       0.000000  SS(1976)=        0.000000  SS(1977)=        0.000000
SS(1978)=       0.000000  SS(1979)=        0.000000  SS(1980)=        0.000000
SS(1981)=       0.000000  SS(1982)=        0.000000  SS(1983)=        0.000000
SS(1984)=       0.000000  SS(1985)=    -1249.999760  SS(1986)=        0.000000
SS(1987)= 199999.812000  SS(1988)=        0.000000  SS(1989)=        0.000000
SS(1990)=  24999.992200  SS(1991)=      624.999760  SS(1992)=        0.000000
SS(1993)=       0.000000  SS(1994)=        0.000000  SS(1995)=        0.000000
SS(1996)=       0.000000 SS(1997)=       0.000000 SS(1998)=       0.000000
SS(1999)=       0.000000 SS(2000)=       0.000000 SS(2001)=       0.000000
SS(2002)=   24999.992200 SS(2003)=       0.000000 SS(2004)=     624.999760
SS(2005)=    2520.832500 SS(2006)=       0.000000 SS(2007)=       0.000000
SS(2008)=   -1249.999760 SS(2009)=       0.000000 SS(2010)=       0.000000
SS(2011)=       0.000000 SS(2012)=       0.000000 SS(2013)=       0.000000
SS(2014)=       0.000000 SS(2015)=       0.000000 SS(2016)=       0.000000
SS(2017)=       0.000000 SS(2018)=    -624.999760 SS(2019)=       0.000000
SS(2020)=     -10.416666 SS(2021)=    2520.832500 SS(2022)=       0.000000
SS(2023)=    -624.999760 SS(2024)=     -10.416666 SS(2025)=       0.000000
SS(2026)=       0.000000 SS(2027)=       0.000000 SS(2028)=       0.000000
SS(2029)=       0.000000 SS(2030)=       0.000000 SS(2031)=       0.000000
SS(2032)=       0.000000 SS(2033)=       0.000000 SS(2034)=       0.000000
SS(2035)=       0.000000 SS(2036)=   -1249.999760 SS(2037)=       0.000000
SS(2038)=  199999.812000 SS(2039)=       0.000000 SS(2040)=       0.000000
SS(2041)=   24999.992200 SS(2042)=     624.999760 SS(2043)=       0.000000
SS(2044)=       0.000000 SS(2045)=       0.000000 SS(2046)=       0.000000
SS(2047)=       0.000000 SS(2048)=       0.000000 SS(2049)=       0.000000
SS(2050)=       0.000000 SS(2051)=       0.000000 SS(2052)=       0.000000
SS(2053)=   24999.992200 SS(2054)=       0.000000 SS(2055)=     624.999760
SS(2056)=    1270.832760 SS(2057)=       0.000000 SS(2058)=       0.000000
SS(2059)=   -1249.999760 SS(2060)=       0.000000 SS(2061)=       0.000000
SS(2062)=       0.000000 SS(2063)=       0.000000 SS(2064)=       0.000000
SS(2065)=       0.000000 SS(2066)=       0.000000 SS(2067)=       0.000000
SS(2068)=       0.000000 SS(2069)=    -624.999760 SS(2070)=       0.000000
SS(2071)=     -10.416666 SS(2072)=    2510.416000 SS(2073)=       0.000000
SS(2074)=    -624.999760 SS(2075)=     -10.416666 SS(2076)=       0.000000
SS(2077)=       0.000000 SS(2078)=       0.000000 SS(2079)=       0.000000
SS(2080)=       0.000000 SS(2081)=       0.000000 SS(2082)=       0.000000
SS(2083)=       0.000000 SS(2084)=       0.000000 SS(2085)=       0.000000
SS(2086)=       0.000000 SS(2087)=   -1249.999760 SS(2088)=       0.000000
SS(2089)=  149999.875000 SS(2090)=    -624.999760 SS(2091)=       0.000000
SS(2092)=   24999.992200 SS(2093)=     624.999760 SS(2094)=       0.000000
SS(2095)=       0.000000 SS(2096)=       0.000000 SS(2097)=       0.000000
SS(2098)=       0.000000 SS(2099)=       0.000000 SS(2100)=       0.000000
SS(2101)=       0.000000 SS(2102)=       0.000000 SS(2103)=       0.000000
SS(2104)=   24999.992200 SS(2105)=       0.000000 SS(2106)=     624.999760
SS(2107)=    1270.832760 SS(2108)=       0.000000 SS(2109)=       0.000000
SS(2110)=       0.000000 SS(2111)=       0.000000 SS(2112)=       0.000000
SS(2113)=       0.000000 SS(2114)=       0.000000 SS(2115)=       0.000000
SS(2116)=       0.000000 SS(2117)=       0.000000 SS(2118)=       0.000000
SS(2119)=       0.000000 SS(2120)=    -624.999760 SS(2121)=       0.000000
SS(2122)=     -10.416666 SS(2123)=    2510.416000 SS(2124)=       0.000000
SS(2125)=       0.000000 SS(2126)=       0.000000 SS(2127)=       0.000000
SS(2128)=       0.000000 SS(2129)=       0.000000 SS(2130)=       0.000000
SS(2131)=       0.000000 SS(2132)=       0.000000 SS(2133)=       0.000000
SS(2134)=       0.000000 SS(2135)=       0.000000 SS(2136)=       0.000000
SS(2137)=       0.000000 SS(2138)=   -1249.999760 SS(2139)=       0.000000
SS(2140)=  149999.875000 SS(2141)=     624.999760 SS(2142)=       0.000000
SS(2143)=       0.000000 SS(2144)=       0.000000 SS(2145)=       0.000000
SS(2146)=       0.000000 SS(2147)=       0.000000 SS(2148)=       0.000000
SS(2149)=       0.000000 SS(2150)=       0.000000 SS(2151)=       0.000000
SS(2152)=       0.000000 SS(2153)=       0.000000 SS(2154)=       0.000000
SS(2155)=   24999.992200 SS(2156)=       0.000000 SS(2157)=     624.999760
SS(2158)=    2520.832500 SS(2159)=       0.000000 SS(2160)=       0.000000
SS(2161)=   -1249.999760 SS(2162)=       0.000000 SS(2163)=       0.000000
SS(2164)=       0.000000 SS(2165)=       0.000000 SS(2166)=       0.000000
SS(2167)=       0.000000 SS(2168)=       0.000000 SS(2169)=       0.000000
SS(2170)=       0.000000 SS(2171)=    -624.999760 SS(2172)=       0.000000
SS(2173)=     -10.416666 SS(2174)=    2520.832500 SS(2175)=       0.000000
SS(2176)=    -624.999760 SS(2177)=     -10.416666 SS(2178)=       0.000000
SS(2179)=       0.000000 SS(2180)=       0.000000 SS(2181)=       0.000000
SS(2182)=       0.000000 SS(2183)=       0.000000 SS(2184)=       0.000000
SS(2185)=      0.000000  SS(2186)=      0.000000  SS(2187)=      0.000000
SS(2188)=      0.000000  SS(2189)=  -1249.999760  SS(2190)=      0.000000
SS(2191)= 199999.812000  SS(2192)=      0.000000  SS(2193)=      0.000000
SS(2194)=  24999.992200  SS(2195)=    624.999760  SS(2196)=      0.000000
SS(2197)=      0.000000  SS(2198)=      0.000000  SS(2199)=      0.000000
SS(2200)=      0.000000  SS(2201)=      0.000000  SS(2202)=      0.000000
SS(2203)=      0.000000  SS(2204)=      0.000000  SS(2205)=      0.000000
SS(2206)=  24999.992200  SS(2207)=      0.000000  SS(2208)=    624.999760
SS(2209)=   2520.832500  SS(2210)=      0.000000  SS(2211)=      0.000000
SS(2212)=  -1249.999760  SS(2213)=      0.000000  SS(2214)=      0.000000
SS(2215)=      0.000000  SS(2216)=      0.000000  SS(2217)=      0.000000
SS(2218)=      0.000000  SS(2219)=      0.000000  SS(2220)=      0.000000
SS(2221)=      0.000000  SS(2222)=   -624.999760  SS(2223)=      0.000000
SS(2224)=    -10.416666  SS(2225)=   2520.832500  SS(2226)=      0.000000
SS(2227)=   -624.999760  SS(2228)=    -10.416666  SS(2229)=      0.000000
SS(2230)=      0.000000  SS(2231)=      0.000000  SS(2232)=      0.000000
SS(2233)=      0.000000  SS(2234)=      0.000000  SS(2235)=      0.000000
SS(2236)=      0.000000  SS(2237)=      0.000000  SS(2238)=      0.000000
SS(2239)=      0.000000  SS(2240)=  -1249.999760  SS(2241)=      0.000000
SS(2242)= 199999.812000  SS(2243)=      0.000000  SS(2244)=      0.000000
SS(2245)=  24999.992200  SS(2246)=    624.999760  SS(2247)=      0.000000
SS(2248)=      0.000000  SS(2249)=      0.000000  SS(2250)=      0.000000
SS(2251)=      0.000000  SS(2252)=      0.000000  SS(2253)=      0.000000
SS(2254)=      0.000000  SS(2255)=      0.000000  SS(2256)=      0.000000
SS(2257)=  24999.992200  SS(2258)=      0.000000  SS(2259)=    624.999760
SS(2260)=   2520.832500  SS(2261)=      0.000000  SS(2262)=      0.000000
SS(2263)=  -1249.999760  SS(2264)=      0.000000  SS(2265)=      0.000000
SS(2266)=      0.000000  SS(2267)=      0.000000  SS(2268)=      0.000000
SS(2269)=      0.000000  SS(2270)=      0.000000  SS(2271)=      0.000000
SS(2272)=      0.000000  SS(2273)=   -624.999760  SS(2274)=      0.000000
SS(2275)=    -10.416666  SS(2276)=   2520.832500  SS(2277)=      0.000000
SS(2278)=   -624.999760  SS(2279)=    -10.416666  SS(2280)=      0.000000
SS(2281)=      0.000000  SS(2282)=      0.000000  SS(2283)=      0.000000
SS(2284)=      0.000000  SS(2285)=      0.000000  SS(2286)=      0.000000
SS(2287)=      0.000000  SS(2288)=      0.000000  SS(2289)=      0.000000
SS(2290)=      0.000000  SS(2291)=  -1249.999760  SS(2292)=      0.000000
SS(2293)= 199999.812000  SS(2294)=      0.000000  SS(2295)=      0.000000
SS(2296)=  24999.992200  SS(2297)=    624.999760  SS(2298)=      0.000000
SS(2299)=      0.000000  SS(2300)=      0.000000  SS(2301)=      0.000000
SS(2302)=      0.000000  SS(2303)=      0.000000  SS(2304)=      0.000000
SS(2305)=      0.000000  SS(2306)=      0.000000  SS(2307)=      0.000000
SS(2308)=  24999.992200  SS(2309)=      0.000000  SS(2310)=    624.999760
SS(2311)=   1270.832760  SS(2312)=      0.000000  SS(2313)=      0.000000
SS(2314)=  -1249.999760  SS(2315)=      0.000000  SS(2316)=      0.000000
SS(2317)=      0.000000  SS(2318)=      0.000000  SS(2319)=      0.000000
SS(2320)=      0.000000  SS(2321)=      0.000000  SS(2322)=      0.000000
SS(2323)=      0.000000  SS(2324)=   -624.999760  SS(2325)=      0.000000
SS(2326)=    -10.416666  SS(2327)=   2510.416000  SS(2328)=      0.000000
SS(2329)=   -624.999760  SS(2330)=    -10.416666  SS(2331)=      0.000000
SS(2332)=      0.000000  SS(2333)=      0.000000  SS(2334)=      0.000000
SS(2335)=      0.000000  SS(2336)=      0.000000  SS(2337)=      0.000000
SS(2338)=      0.000000  SS(2339)=      0.000000  SS(2340)=      0.000000
SS(2341)=      0.000000  SS(2342)=  -1249.999760  SS(2343)=      0.000000
SS(2344)= 149999.875000  SS(2345)=   -624.999760  SS(2346)=      0.000000
SS(2347)=  24999.992200  SS(2348)=    624.999760  SS(2349)=      0.000000
SS(2350)=      0.000000  SS(2351)=      0.000000  SS(2352)=      0.000000
SS(2353)=      0.000000  SS(2354)=      0.000000  SS(2355)=      0.000000
SS(2356)=      0.000000  SS(2357)=      0.000000  SS(2358)=      0.000000
SS(2359)=  24999.992200  SS(2360)=      0.000000  SS(2361)=    624.999760
SS(2362)=   1270.832760  SS(2363)=      0.000000  SS(2364)=      0.000000
SS(2365)=      0.000000  SS(2366)=      0.000000  SS(2367)=      0.000000
SS(2368)=      0.000000  SS(2369)=      0.000000  SS(2370)=      0.000000
SS(2371)=      0.000000  SS(2372)=      0.000000  SS(2373)=      0.000000
SS(2374)=        0.000000 SS(2375)=     -624.999760 SS(2376)=        0.000000
SS(2377)=      -10.416666 SS(2378)=     2510.416000 SS(2379)=        0.000000
SS(2380)=        0.000000 SS(2381)=        0.000000 SS(2382)=        0.000000
SS(2383)=        0.000000 SS(2384)=        0.000000 SS(2385)=        0.000000
SS(2386)=        0.000000 SS(2387)=        0.000000 SS(2388)=        0.000000
SS(2389)=        0.000000 SS(2390)=        0.000000 SS(2391)=        0.000000
SS(2392)=        0.000000 SS(2393)=    -1249.999760 SS(2394)=        0.000000
SS(2395)=   149999.875000 SS(2396)=      624.999760 SS(2397)=        0.000000
SS(2398)=        0.000000 SS(2399)=        0.000000 SS(2400)=        0.000000
SS(2401)=        0.000000 SS(2402)=        0.000000 SS(2403)=        0.000000
SS(2404)=        0.000000 SS(2405)=        0.000000 SS(2406)=        0.000000
SS(2407)=        0.000000 SS(2408)=        0.000000 SS(2409)=        0.000000
SS(2410)=    24999.992200 SS(2411)=        0.000000 SS(2412)=      624.999760
SS(2413)=     2520.832500 SS(2414)=        0.000000 SS(2415)=        0.000000
SS(2416)=    -1249.999760 SS(2417)=        0.000000 SS(2418)=        0.000000
SS(2419)=        0.000000 SS(2420)=        0.000000 SS(2421)=        0.000000
SS(2422)=        0.000000 SS(2423)=        0.000000 SS(2424)=        0.000000
SS(2425)=        0.000000 SS(2426)=     -624.999760 SS(2427)=        0.000000
SS(2428)=      -10.416666 SS(2429)=     2520.832500 SS(2430)=        0.000000
SS(2431)=     -624.999760 SS(2432)=      -10.416666 SS(2433)=        0.000000
SS(2434)=        0.000000 SS(2435)=        0.000000 SS(2436)=        0.000000
SS(2437)=        0.000000 SS(2438)=        0.000000 SS(2439)=        0.000000
SS(2440)=        0.000000 SS(2441)=        0.000000 SS(2442)=        0.000000
SS(2443)=        0.000000 SS(2444)=    -1249.999760 SS(2445)=        0.000000
SS(2446)=   199999.812000 SS(2447)=        0.000000 SS(2448)=        0.000000
SS(2449)=    24999.992200 SS(2450)=      624.999760 SS(2451)=        0.000000
SS(2452)=        0.000000 SS(2453)=        0.000000 SS(2454)=        0.000000
SS(2455)=        0.000000 SS(2456)=        0.000000 SS(2457)=        0.000000
SS(2458)=        0.000000 SS(2459)=        0.000000 SS(2460)=        0.000000
SS(2461)=    24999.992200 SS(2462)=        0.000000 SS(2463)=      624.999760
SS(2464)=     2520.832500 SS(2465)=        0.000000 SS(2466)=        0.000000
SS(2467)=    -1249.999760 SS(2468)=        0.000000 SS(2469)=        0.000000
SS(2470)=        0.000000 SS(2471)=        0.000000 SS(2472)=        0.000000
SS(2473)=        0.000000 SS(2474)=        0.000000 SS(2475)=        0.000000
SS(2476)=        0.000000 SS(2477)=     -624.999760 SS(2478)=        0.000000
SS(2479)=      -10.416666 SS(2480)=     2520.832500 SS(2481)=        0.000000
SS(2482)=     -624.999760 SS(2483)=      -10.416666 SS(2484)=        0.000000
SS(2485)=        0.000000 SS(2486)=        0.000000 SS(2487)=        0.000000
SS(2488)=        0.000000 SS(2489)=        0.000000 SS(2490)=        0.000000
SS(2491)=        0.000000 SS(2492)=        0.000000 SS(2493)=        0.000000
SS(2494)=        0.000000 SS(2495)=    -1249.999760 SS(2496)=        0.000000
SS(2497)=   199999.812000 SS(2498)=        0.000000 SS(2499)=        0.000000
SS(2500)=    24999.992200 SS(2501)=      624.999760 SS(2502)=        0.000000
SS(2503)=        0.000000 SS(2504)=        0.000000 SS(2505)=        0.000000
SS(2506)=        0.000000 SS(2507)=        0.000000 SS(2508)=        0.000000
SS(2509)=        0.000000 SS(2510)=        0.000000 SS(2511)=        0.000000
SS(2512)=    24999.992200 SS(2513)=        0.000000 SS(2514)=      624.999760
SS(2515)=     2520.832500 SS(2516)=        0.000000 SS(2517)=        0.000000
SS(2518)=    -1249.999760 SS(2519)=        0.000000 SS(2520)=        0.000000
SS(2521)=        0.000000 SS(2522)=        0.000000 SS(2523)=        0.000000
SS(2524)=        0.000000 SS(2525)=        0.000000 SS(2526)=        0.000000
SS(2527)=        0.000000 SS(2528)=     -624.999760 SS(2529)=        0.000000
SS(2530)=      -10.416666 SS(2531)=     2520.832500 SS(2532)=        0.000000
SS(2533)=     -624.999760 SS(2534)=      -10.416666 SS(2535)=        0.000000
SS(2536)=        0.000000 SS(2537)=        0.000000 SS(2538)=        0.000000
SS(2539)=        0.000000 SS(2540)=        0.000000 SS(2541)=        0.000000
SS(2542)=        0.000000 SS(2543)=        0.000000 SS(2544)=        0.000000
SS(2545)=        0.000000 SS(2546)=    -1249.999760 SS(2547)=        0.000000
SS(2548)=   199999.812000 SS(2549)=        0.000000 SS(2550)=        0.000000
SS(2551)=    24999.992200 SS(2552)=      624.999760 SS(2553)=        0.000000
SS(2554)=        0.000000 SS(2555)=        0.000000 SS(2556)=        0.000000
SS(2557)=        0.000000 SS(2558)=        0.000000 SS(2559)=        0.000000
SS(2560)=        0.000000 SS(2561)=        0.000000 SS(2562)=        0.000000
SS(2563)=  24999.992200  SS(2564)=      0.000000  SS(2565)=     624.999760
SS(2566)=   1270.832760  SS(2567)=      0.000000  SS(2568)=       0.000000
SS(2569)=  -1249.999760  SS(2570)=      0.000000  SS(2571)=       0.000000
SS(2572)=      0.000000  SS(2573)=      0.000000  SS(2574)=       0.000000
SS(2575)=      0.000000  SS(2576)=      0.000000  SS(2577)=       0.000000
SS(2578)=      0.000000  SS(2579)=   -624.999760  SS(2580)=       0.000000
SS(2581)=    -10.416666  SS(2582)=   2510.416000  SS(2583)=       0.000000
SS(2584)=   -624.999760  SS(2585)=    -10.416666  SS(2586)=       0.000000
SS(2587)=      0.000000  SS(2588)=      0.000000  SS(2589)=       0.000000
SS(2590)=      0.000000  SS(2591)=      0.000000  SS(2592)=       0.000000
SS(2593)=      0.000000  SS(2594)=      0.000000  SS(2595)=       0.000000
SS(2596)=      0.000000  SS(2597)=  -1249.999760  SS(2598)=       0.000000
SS(2599)= 149999.875000  SS(2600)=   -624.999760  SS(2601)=       0.000000
SS(2602)=  24999.992200  SS(2603)=    624.999760  SS(2604)=       0.000000
SS(2605)=      0.000000  SS(2606)=      0.000000  SS(2607)=       0.000000
SS(2608)=      0.000000  SS(2609)=      0.000000  SS(2610)=       0.000000
SS(2611)=      0.000000  SS(2612)=      0.000000  SS(2613)=       0.000000
SS(2614)=  24999.992200  SS(2615)=      0.000000  SS(2616)=     624.999760
SS(2617)=   1270.832760  SS(2618)=      0.000000  SS(2619)=       0.000000
SS(2620)=      0.000000  SS(2621)=      0.000000  SS(2622)=       0.000000
SS(2623)=      0.000000  SS(2624)=      0.000000  SS(2625)=       0.000000
SS(2626)=      0.000000  SS(2627)=      0.000000  SS(2628)=       0.000000
SS(2629)=      0.000000  SS(2630)=   -624.999760  SS(2631)=       0.000000
SS(2632)=    -10.416666  SS(2633)=   2510.416000  SS(2634)=       0.000000
SS(2635)=      0.000000  SS(2636)=      0.000000  SS(2637)=       0.000000
SS(2638)=      0.000000  SS(2639)=      0.000000  SS(2640)=       0.000000
SS(2641)=      0.000000  SS(2642)=      0.000000  SS(2643)=       0.000000
SS(2644)=      0.000000  SS(2645)=      0.000000  SS(2646)=       0.000000
SS(2647)=      0.000000  SS(2648)=  -1249.999760  SS(2649)=       0.000000
SS(2650)= 149999.875000  SS(2651)=    624.999760  SS(2652)=       0.000000
SS(2653)=      0.000000  SS(2654)=      0.000000  SS(2655)=       0.000000
SS(2656)=      0.000000  SS(2657)=      0.000000  SS(2658)=       0.000000
SS(2659)=      0.000000  SS(2660)=      0.000000  SS(2661)=       0.000000
SS(2662)=      0.000000  SS(2663)=      0.000000  SS(2664)=       0.000000
SS(2665)=  24999.992200  SS(2666)=      0.000000  SS(2667)=     624.999760
SS(2668)=   2520.832500  SS(2669)=      0.000000  SS(2670)=       0.000000
SS(2671)=  -1249.999760  SS(2672)=      0.000000  SS(2673)=       0.000000
SS(2674)=      0.000000  SS(2675)=      0.000000  SS(2676)=       0.000000
SS(2677)=      0.000000  SS(2678)=      0.000000  SS(2679)=       0.000000
SS(2680)=      0.000000  SS(2681)=   -624.999760  SS(2682)=       0.000000
SS(2683)=    -10.416666  SS(2684)=   2520.832500  SS(2685)=       0.000000
SS(2686)=   -624.999760  SS(2687)=    -10.416666  SS(2688)=       0.000000
SS(2689)=      0.000000  SS(2690)=      0.000000  SS(2691)=       0.000000
SS(2692)=      0.000000  SS(2693)=      0.000000  SS(2694)=       0.000000
SS(2695)=      0.000000  SS(2696)=      0.000000  SS(2697)=       0.000000
SS(2698)=      0.000000  SS(2699)=  -1249.999760  SS(2700)=       0.000000
SS(2701)= 199999.812000  SS(2702)=      0.000000  SS(2703)=       0.000000
SS(2704)=  24999.992200  SS(2705)=    624.999760  SS(2706)=       0.000000
SS(2707)=      0.000000  SS(2708)=      0.000000  SS(2709)=       0.000000
SS(2710)=      0.000000  SS(2711)=      0.000000  SS(2712)=       0.000000
SS(2713)=      0.000000  SS(2714)=      0.000000  SS(2715)=       0.000000
SS(2716)=  24999.992200  SS(2717)=      0.000000  SS(2718)=     624.999760
SS(2719)=   2520.832500  SS(2720)=      0.000000  SS(2721)=       0.000000
SS(2722)=  -1249.999760  SS(2723)=      0.000000  SS(2724)=       0.000000
SS(2725)=      0.000000  SS(2726)=      0.000000  SS(2727)=       0.000000
SS(2728)=      0.000000  SS(2729)=      0.000000  SS(2730)=       0.000000
SS(2731)=      0.000000  SS(2732)=   -624.999760  SS(2733)=       0.000000
SS(2734)=    -10.416666  SS(2735)=   2520.832500  SS(2736)=       0.000000
SS(2737)=   -624.999760  SS(2738)=    -10.416666  SS(2739)=       0.000000
SS(2740)=      0.000000  SS(2741)=      0.000000  SS(2742)=       0.000000
SS(2743)=      0.000000  SS(2744)=      0.000000  SS(2745)=       0.000000
SS(2746)=      0.000000  SS(2747)=      0.000000  SS(2748)=       0.000000
SS(2749)=      0.000000  SS(2750)=  -1249.999760  SS(2751)=       0.000000
SS(2752)= 199999.812000  SS(2753)=      0.000000  SS(2754)=      0.000000
SS(2755)=  24999.992200  SS(2756)=    624.999760  SS(2757)=      0.000000
SS(2758)=      0.000000  SS(2759)=      0.000000  SS(2760)=      0.000000
SS(2761)=      0.000000  SS(2762)=      0.000000  SS(2763)=      0.000000
SS(2764)=      0.000000  SS(2765)=      0.000000  SS(2766)=      0.000000
SS(2767)=  24999.992200  SS(2768)=      0.000000  SS(2769)=    624.999760
SS(2770)=   2520.832500  SS(2771)=      0.000000  SS(2772)=      0.000000
SS(2773)=  -1249.999760  SS(2774)=      0.000000  SS(2775)=      0.000000
SS(2776)=      0.000000  SS(2777)=      0.000000  SS(2778)=      0.000000
SS(2779)=      0.000000  SS(2780)=      0.000000  SS(2781)=      0.000000
SS(2782)=      0.000000  SS(2783)=   -624.999760  SS(2784)=      0.000000
SS(2785)=    -10.416666  SS(2786)=   2520.832500  SS(2787)=      0.000000
SS(2788)=   -624.999760  SS(2789)=    -10.416666  SS(2790)=      0.000000
SS(2791)=      0.000000  SS(2792)=      0.000000  SS(2793)=      0.000000
SS(2794)=      0.000000  SS(2795)=      0.000000  SS(2796)=      0.000000
SS(2797)=      0.000000  SS(2798)=      0.000000  SS(2799)=      0.000000
SS(2800)=      0.000000  SS(2801)=  -1249.999760  SS(2802)=      0.000000
SS(2803)= 199999.812000  SS(2804)=      0.000000  SS(2805)=      0.000000
SS(2806)=  24999.992200  SS(2807)=    624.999760  SS(2808)=      0.000000
SS(2809)=      0.000000  SS(2810)=      0.000000  SS(2811)=      0.000000
SS(2812)=      0.000000  SS(2813)=      0.000000  SS(2814)=      0.000000
SS(2815)=      0.000000  SS(2816)=      0.000000  SS(2817)=      0.000000
SS(2818)=  24999.992200  SS(2819)=      0.000000  SS(2820)=    624.999760
SS(2821)=   1270.832760  SS(2822)=      0.000000  SS(2823)=      0.000000
SS(2824)=  -1249.999760  SS(2825)=      0.000000  SS(2826)=      0.000000
SS(2827)=      0.000000  SS(2828)=      0.000000  SS(2829)=      0.000000
SS(2830)=      0.000000  SS(2831)=      0.000000  SS(2832)=      0.000000
SS(2833)=      0.000000  SS(2834)=   -624.999760  SS(2835)=      0.000000
SS(2836)=    -10.416666  SS(2837)=   2510.416000  SS(2838)=      0.000000
SS(2839)=   -624.999760  SS(2840)=    -10.416666  SS(2841)=      0.000000
SS(2842)=      0.000000  SS(2843)=      0.000000  SS(2844)=      0.000000
SS(2845)=      0.000000  SS(2846)=      0.000000  SS(2847)=      0.000000
SS(2848)=      0.000000  SS(2849)=      0.000000  SS(2850)=      0.000000
SS(2851)=      0.000000  SS(2852)=  -1249.999760  SS(2853)=      0.000000
SS(2854)= 149999.875000  SS(2855)=   -624.999760  SS(2856)=      0.000000
SS(2857)=  24999.992200  SS(2858)=    624.999760  SS(2859)=      0.000000
SS(2860)=      0.000000  SS(2861)=      0.000000  SS(2862)=      0.000000
SS(2863)=      0.000000  SS(2864)=      0.000000  SS(2865)=      0.000000
SS(2866)=      0.000000  SS(2867)=      0.000000  SS(2868)=      0.000000
SS(2869)=  24999.992200  SS(2870)=      0.000000  SS(2871)=    624.999760
SS(2872)=   1270.832760  SS(2873)=      0.000000  SS(2874)=      0.000000
SS(2875)=      0.000000  SS(2876)=      0.000000  SS(2877)=      0.000000
SS(2878)=      0.000000  SS(2879)=      0.000000  SS(2880)=      0.000000
SS(2881)=      0.000000  SS(2882)=      0.000000  SS(2883)=      0.000000
SS(2884)=      0.000000  SS(2885)=   -624.999760  SS(2886)=      0.000000
SS(2887)=    -10.416666  SS(2888)=   2510.416000  SS(2889)=      0.000000
SS(2890)=      0.000000  SS(2891)=      0.000000  SS(2892)=      0.000000
SS(2893)=      0.000000  SS(2894)=      0.000000  SS(2895)=      0.000000
SS(2896)=      0.000000  SS(2897)=      0.000000  SS(2898)=      0.000000
SS(2899)=      0.000000  SS(2900)=      0.000000  SS(2901)=      0.000000
SS(2902)=      0.000000  SS(2903)=  -1249.999760  SS(2904)=      0.000000
SS(2905)= 149999.875000  SS(2906)=    624.999760  SS(2907)=      0.000000
SS(2908)=      0.000000  SS(2909)=      0.000000  SS(2910)=      0.000000
SS(2911)=      0.000000  SS(2912)=      0.000000  SS(2913)=      0.000000
SS(2914)=      0.000000  SS(2915)=      0.000000  SS(2916)=      0.000000
SS(2917)=      0.000000  SS(2918)=      0.000000  SS(2919)=      0.000000
SS(2920)=  24999.992200  SS(2921)=      0.000000  SS(2922)=    624.999760
SS(2923)=   2520.832500  SS(2924)=      0.000000  SS(2925)=      0.000000
SS(2926)=  -1249.999760  SS(2927)=      0.000000  SS(2928)=      0.000000
SS(2929)=      0.000000  SS(2930)=      0.000000  SS(2931)=      0.000000
SS(2932)=      0.000000  SS(2933)=      0.000000  SS(2934)=      0.000000
SS(2935)=      0.000000  SS(2936)=   -624.999760  SS(2937)=      0.000000
SS(2938)=    -10.416666  SS(2939)=   2520.832500  SS(2940)=      0.000000
SS(2941)=   -624.999760  SS(2942)=     -10.416666  SS(2943)=       0.000000
SS(2944)=      0.000000  SS(2945)=       0.000000  SS(2946)=       0.000000
SS(2947)=      0.000000  SS(2948)=       0.000000  SS(2949)=       0.000000
SS(2950)=      0.000000  SS(2951)=       0.000000  SS(2952)=       0.000000
SS(2953)=      0.000000  SS(2954)=   -1249.999760  SS(2955)=       0.000000
SS(2956)= 199999.812000  SS(2957)=       0.000000  SS(2958)=       0.000000
SS(2959)=  24999.992200  SS(2960)=     624.999760  SS(2961)=       0.000000
SS(2962)=      0.000000  SS(2963)=       0.000000  SS(2964)=       0.000000
SS(2965)=      0.000000  SS(2966)=       0.000000  SS(2967)=       0.000000
SS(2968)=      0.000000  SS(2969)=       0.000000  SS(2970)=       0.000000
SS(2971)=  24999.992200  SS(2972)=       0.000000  SS(2973)=     624.999760
SS(2974)=   2520.832500  SS(2975)=       0.000000  SS(2976)=       0.000000
SS(2977)=  -1249.999760  SS(2978)=       0.000000  SS(2979)=       0.000000
SS(2980)=      0.000000  SS(2981)=       0.000000  SS(2982)=       0.000000
SS(2983)=      0.000000  SS(2984)=       0.000000  SS(2985)=       0.000000
SS(2986)=      0.000000  SS(2987)=    -624.999760  SS(2988)=       0.000000
SS(2989)=    -10.416666  SS(2990)=    2520.832500  SS(2991)=       0.000000
SS(2992)=   -624.999760  SS(2993)=     -10.416666  SS(2994)=       0.000000
SS(2995)=      0.000000  SS(2996)=       0.000000  SS(2997)=       0.000000
SS(2998)=      0.000000  SS(2999)=       0.000000  SS(3000)=       0.000000
SS(3001)=      0.000000  SS(3002)=       0.000000  SS(3003)=       0.000000
SS(3004)=      0.000000  SS(3005)=   -1249.999760  SS(3006)=       0.000000
SS(3007)= 199999.812000  SS(3008)=       0.000000  SS(3009)=       0.000000
SS(3010)=  24999.992200  SS(3011)=     624.999760  SS(3012)=       0.000000
SS(3013)=      0.000000  SS(3014)=       0.000000  SS(3015)=       0.000000
SS(3016)=      0.000000  SS(3017)=       0.000000  SS(3018)=       0.000000
SS(3019)=      0.000000  SS(3020)=       0.000000  SS(3021)=       0.000000
SS(3022)=  24999.992200  SS(3023)=       0.000000  SS(3024)=     624.999760
SS(3025)=   2520.832500  SS(3026)=       0.000000  SS(3027)=       0.000000
SS(3028)=  -1249.999760  SS(3029)=       0.000000  SS(3030)=       0.000000
SS(3031)=      0.000000  SS(3032)=       0.000000  SS(3033)=       0.000000
SS(3034)=      0.000000  SS(3035)=       0.000000  SS(3036)=       0.000000
SS(3037)=      0.000000  SS(3038)=    -624.999760  SS(3039)=       0.000000
SS(3040)=    -10.416666  SS(3041)=    2520.832500  SS(3042)=       0.000000
SS(3043)=   -624.999760  SS(3044)=     -10.416666  SS(3045)=       0.000000
SS(3046)=      0.000000  SS(3047)=       0.000000  SS(3048)=       0.000000
SS(3049)=      0.000000  SS(3050)=       0.000000  SS(3051)=       0.000000
SS(3052)=      0.000000  SS(3053)=       0.000000  SS(3054)=       0.000000
SS(3055)=      0.000000  SS(3056)=   -1249.999760  SS(3057)=       0.000000
SS(3058)= 199999.812000  SS(3059)=       0.000000  SS(3060)=       0.000000
SS(3061)=  24999.992200  SS(3062)=     624.999760  SS(3063)=       0.000000
SS(3064)=      0.000000  SS(3065)=       0.000000  SS(3066)=       0.000000
SS(3067)=      0.000000  SS(3068)=       0.000000  SS(3069)=       0.000000
SS(3070)=      0.000000  SS(3071)=       0.000000  SS(3072)=       0.000000
SS(3073)=  24999.992200  SS(3074)=       0.000000  SS(3075)=     624.999760
SS(3076)=   1270.832760  SS(3077)=       0.000000  SS(3078)=       0.000000
SS(3079)=  -1249.999760  SS(3080)=       0.000000  SS(3081)=       0.000000
SS(3082)=      0.000000  SS(3083)=       0.000000  SS(3084)=       0.000000
SS(3085)=      0.000000  SS(3086)=       0.000000  SS(3087)=       0.000000
SS(3088)=      0.000000  SS(3089)=    -624.999760  SS(3090)=       0.000000
SS(3091)=    -10.416666  SS(3092)=    2510.416000  SS(3093)=       0.000000
SS(3094)=   -624.999760  SS(3095)=     -10.416666  SS(3096)=       0.000000
SS(3097)=      0.000000  SS(3098)=       0.000000  SS(3099)=       0.000000
SS(3100)=      0.000000  SS(3101)=       0.000000  SS(3102)=       0.000000
SS(3103)=      0.000000  SS(3104)=       0.000000  SS(3105)=       0.000000
SS(3106)=      0.000000  SS(3107)=   -1249.999760  SS(3108)=       0.000000
SS(3109)= 149999.875000  SS(3110)=    -624.999760  SS(3111)=       0.000000
SS(3112)=  24999.992200  SS(3113)=     624.999760  SS(3114)=       0.000000
SS(3115)=      0.000000  SS(3116)=       0.000000  SS(3117)=       0.000000
SS(3118)=      0.000000  SS(3119)=       0.000000  SS(3120)=       0.000000
SS(3121)=      0.000000  SS(3122)=       0.000000  SS(3123)=       0.000000
SS(3124)=  24999.992200  SS(3125)=       0.000000  SS(3126)=     624.999760
SS(3127)=   1270.832760  SS(3128)=       0.000000  SS(3129)=       0.000000
SS(3130)=      0.000000  SS(3131)=      0.000000  SS(3132)=      0.000000
SS(3133)=      0.000000  SS(3134)=      0.000000  SS(3135)=      0.000000
SS(3136)=      0.000000  SS(3137)=      0.000000  SS(3138)=      0.000000
SS(3139)=      0.000000  SS(3140)=   -624.999760  SS(3141)=      0.000000
SS(3142)=    -10.416666  SS(3143)=   2510.416000  SS(3144)=      0.000000
SS(3145)=      0.000000  SS(3146)=      0.000000  SS(3147)=      0.000000
SS(3148)=      0.000000  SS(3149)=      0.000000  SS(3150)=      0.000000
SS(3151)=      0.000000  SS(3152)=      0.000000  SS(3153)=      0.000000
SS(3154)=      0.000000  SS(3155)=      0.000000  SS(3156)=      0.000000
SS(3157)=      0.000000  SS(3158)=  -1249.999760  SS(3159)=      0.000000
SS(3160)= 149999.875000  SS(3161)=    624.999760  SS(3162)=      0.000000
SS(3163)=      0.000000  SS(3164)=      0.000000  SS(3165)=      0.000000
SS(3166)=      0.000000  SS(3167)=      0.000000  SS(3168)=      0.000000
SS(3169)=      0.000000  SS(3170)=      0.000000  SS(3171)=      0.000000
SS(3172)=      0.000000  SS(3173)=      0.000000  SS(3174)=      0.000000
SS(3175)=  24999.992200  SS(3176)=      0.000000  SS(3177)=    624.999760
SS(3178)=   2520.832500  SS(3179)=      0.000000  SS(3180)=      0.000000
SS(3181)=  -1249.999760  SS(3182)=      0.000000  SS(3183)=      0.000000
SS(3184)=      0.000000  SS(3185)=      0.000000  SS(3186)=      0.000000
SS(3187)=      0.000000  SS(3188)=      0.000000  SS(3189)=      0.000000
SS(3190)=      0.000000  SS(3191)=   -624.999760  SS(3192)=      0.000000
SS(3193)=    -10.416666  SS(3194)=   2520.832500  SS(3195)=      0.000000
SS(3196)=   -624.999760  SS(3197)=    -10.416666  SS(3198)=      0.000000
SS(3199)=      0.000000  SS(3200)=      0.000000  SS(3201)=      0.000000
SS(3202)=      0.000000  SS(3203)=      0.000000  SS(3204)=      0.000000
SS(3205)=      0.000000  SS(3206)=      0.000000  SS(3207)=      0.000000
SS(3208)=      0.000000  SS(3209)=  -1249.999760  SS(3210)=      0.000000
SS(3211)= 199999.812000  SS(3212)=      0.000000  SS(3213)=      0.000000
SS(3214)=  24999.992200  SS(3215)=    624.999760  SS(3216)=      0.000000
SS(3217)=      0.000000  SS(3218)=      0.000000  SS(3219)=      0.000000
SS(3220)=      0.000000  SS(3221)=      0.000000  SS(3222)=      0.000000
SS(3223)=      0.000000  SS(3224)=      0.000000  SS(3225)=      0.000000
SS(3226)=  24999.992200  SS(3227)=      0.000000  SS(3228)=    624.999760
SS(3229)=   2520.832500  SS(3230)=      0.000000  SS(3231)=      0.000000
SS(3232)=  -1249.999760  SS(3233)=      0.000000  SS(3234)=      0.000000
SS(3235)=      0.000000  SS(3236)=      0.000000  SS(3237)=      0.000000
SS(3238)=      0.000000  SS(3239)=      0.000000  SS(3240)=      0.000000
SS(3241)=      0.000000  SS(3242)=   -624.999760  SS(3243)=      0.000000
SS(3244)=    -10.416666  SS(3245)=   2520.832500  SS(3246)=      0.000000
SS(3247)=   -624.999760  SS(3248)=    -10.416666  SS(3249)=      0.000000
SS(3250)=      0.000000  SS(3251)=      0.000000  SS(3252)=      0.000000
SS(3253)=      0.000000  SS(3254)=      0.000000  SS(3255)=      0.000000
SS(3256)=      0.000000  SS(3257)=      0.000000  SS(3258)=      0.000000
SS(3259)=      0.000000  SS(3260)=  -1249.999760  SS(3261)=      0.000000
SS(3262)= 199999.812000  SS(3263)=      0.000000  SS(3264)=      0.000000
SS(3265)=  24999.992200  SS(3266)=    624.999760  SS(3267)=      0.000000
SS(3268)=      0.000000  SS(3269)=      0.000000  SS(3270)=      0.000000
SS(3271)=      0.000000  SS(3272)=      0.000000  SS(3273)=      0.000000
SS(3274)=      0.000000  SS(3275)=      0.000000  SS(3276)=      0.000000
SS(3277)=  24999.992200  SS(3278)=      0.000000  SS(3279)=    624.999760
SS(3280)=   2520.832500  SS(3281)=      0.000000  SS(3282)=      0.000000
SS(3283)=  -1249.999760  SS(3284)=      0.000000  SS(3285)=      0.000000
SS(3286)=      0.000000  SS(3287)=      0.000000  SS(3288)=      0.000000
SS(3289)=      0.000000  SS(3290)=      0.000000  SS(3291)=      0.000000
SS(3292)=      0.000000  SS(3293)=   -624.999760  SS(3294)=      0.000000
SS(3295)=    -10.416666  SS(3296)=   2520.832500  SS(3297)=      0.000000
SS(3298)=   -624.999760  SS(3299)=    -10.416666  SS(3300)=      0.000000
SS(3301)=      0.000000  SS(3302)=      0.000000  SS(3303)=      0.000000
SS(3304)=      0.000000  SS(3305)=      0.000000  SS(3306)=      0.000000
SS(3307)=      0.000000  SS(3308)=      0.000000  SS(3309)=      0.000000
SS(3310)=      0.000000  SS(3311)=  -1249.999760  SS(3312)=      0.000000
SS(3313)= 199999.812000  SS(3314)=      0.000000  SS(3315)=      0.000000
SS(3316)=  24999.992200  SS(3317)=    624.999760  SS(3318)=      0.000000
SS(3319)=       0.000000 SS(3320)=        0.000000 SS(3321)=        0.000000
SS(3322)=       0.000000 SS(3323)=        0.000000 SS(3324)=        0.000000
SS(3325)=       0.000000 SS(3326)=        0.000000 SS(3327)=        0.000000
SS(3328)=   24999.992200 SS(3329)=        0.000000 SS(3330)=      624.999760
SS(3331)=    1270.832760 SS(3332)=        0.000000 SS(3333)=        0.000000
SS(3334)=   -1249.999760 SS(3335)=        0.000000 SS(3336)=        0.000000
SS(3337)=       0.000000 SS(3338)=        0.000000 SS(3339)=        0.000000
SS(3340)=       0.000000 SS(3341)=        0.000000 SS(3342)=        0.000000
SS(3343)=       0.000000 SS(3344)=     -624.999760 SS(3345)=        0.000000
SS(3346)=     -10.416666 SS(3347)=     2510.416000 SS(3348)=        0.000000
SS(3349)=    -624.999760 SS(3350)=      -10.416666 SS(3351)=        0.000000
SS(3352)=       0.000000 SS(3353)=        0.000000 SS(3354)=        0.000000
SS(3355)=       0.000000 SS(3356)=        0.000000 SS(3357)=        0.000000
SS(3358)=       0.000000 SS(3359)=        0.000000 SS(3360)=        0.000000
SS(3361)=       0.000000 SS(3362)=    -1249.999760 SS(3363)=        0.000000
SS(3364)=  149999.875000 SS(3365)=     -624.999760 SS(3366)=        0.000000
SS(3367)=   24999.992200 SS(3368)=      624.999760 SS(3369)=        0.000000
SS(3370)=       0.000000 SS(3371)=        0.000000 SS(3372)=        0.000000
SS(3373)=       0.000000 SS(3374)=        0.000000 SS(3375)=        0.000000
SS(3376)=       0.000000 SS(3377)=        0.000000 SS(3378)=        0.000000
SS(3379)=   24999.992200 SS(3380)=        0.000000 SS(3381)=      624.999760
SS(3382)=    1270.832760 SS(3383)=        0.000000 SS(3384)=        0.000000
SS(3385)=       0.000000 SS(3386)=        0.000000 SS(3387)=        0.000000
SS(3388)=       0.000000 SS(3389)=        0.000000 SS(3390)=        0.000000
SS(3391)=       0.000000 SS(3392)=        0.000000 SS(3393)=        0.000000
SS(3394)=       0.000000 SS(3395)=     -624.999760 SS(3396)=        0.000000
SS(3397)=     -10.416666 SS(3398)=     2510.416000 SS(3399)=        0.000000
SS(3400)=       0.000000 SS(3401)=        0.000000 SS(3402)=        0.000000
SS(3403)=       0.000000 SS(3404)=        0.000000 SS(3405)=        0.000000
SS(3406)=       0.000000 SS(3407)=        0.000000 SS(3408)=        0.000000
SS(3409)=       0.000000 SS(3410)=        0.000000 SS(3411)=        0.000000
SS(3412)=       0.000000 SS(3413)=    -1249.999760 SS(3414)=        0.000000
SS(3415)=  149999.875000 SS(3416)=      624.999760 SS(3417)=        0.000000
SS(3418)=       0.000000 SS(3419)=        0.000000 SS(3420)=        0.000000
SS(3421)=       0.000000 SS(3422)=        0.000000 SS(3423)=        0.000000
SS(3424)=       0.000000 SS(3425)=        0.000000 SS(3426)=        0.000000
SS(3427)=       0.000000 SS(3428)=        0.000000 SS(3429)=        0.000000
SS(3430)=   24999.992200 SS(3431)=        0.000000 SS(3432)=      624.999760
SS(3433)=    2520.832500 SS(3434)=        0.000000 SS(3435)=        0.000000
SS(3436)=   -1249.999760 SS(3437)=        0.000000 SS(3438)=        0.000000
SS(3439)=       0.000000 SS(3440)=        0.000000 SS(3441)=        0.000000
SS(3442)=       0.000000 SS(3443)=        0.000000 SS(3444)=        0.000000
SS(3445)=       0.000000 SS(3446)=     -624.999760 SS(3447)=        0.000000
SS(3448)=     -10.416666 SS(3449)=     2520.832500 SS(3450)=        0.000000
SS(3451)=    -624.999760 SS(3452)=      -10.416666 SS(3453)=        0.000000
SS(3454)=       0.000000 SS(3455)=        0.000000 SS(3456)=        0.000000
SS(3457)=       0.000000 SS(3458)=        0.000000 SS(3459)=        0.000000
SS(3460)=       0.000000 SS(3461)=        0.000000 SS(3462)=        0.000000
SS(3463)=       0.000000 SS(3464)=    -1249.999760 SS(3465)=        0.000000
SS(3466)=  199999.812000 SS(3467)=        0.000000 SS(3468)=        0.000000
SS(3469)=   24999.992200 SS(3470)=      624.999760 SS(3471)=        0.000000
SS(3472)=       0.000000 SS(3473)=        0.000000 SS(3474)=        0.000000
SS(3475)=       0.000000 SS(3476)=        0.000000 SS(3477)=        0.000000
SS(3478)=       0.000000 SS(3479)=        0.000000 SS(3480)=        0.000000
SS(3481)=   24999.992200 SS(3482)=        0.000000 SS(3483)=      624.999760
SS(3484)=    2520.832500 SS(3485)=        0.000000 SS(3486)=        0.000000
SS(3487)=   -1249.999760 SS(3488)=        0.000000 SS(3489)=        0.000000
SS(3490)=       0.000000 SS(3491)=        0.000000 SS(3492)=        0.000000
SS(3493)=       0.000000 SS(3494)=        0.000000 SS(3495)=        0.000000
SS(3496)=       0.000000 SS(3497)=     -624.999760 SS(3498)=        0.000000
SS(3499)=     -10.416666 SS(3500)=     2520.832500 SS(3501)=        0.000000
SS(3502)=    -624.999760 SS(3503)=      -10.416666 SS(3504)=        0.000000
SS(3505)=       0.000000 SS(3506)=        0.000000 SS(3507)=        0.000000
SS(3508)=      0.000000  SS(3509)=      0.000000  SS(3510)=      0.000000
SS(3511)=      0.000000  SS(3512)=      0.000000  SS(3513)=      0.000000
SS(3514)=      0.000000  SS(3515)=  -1249.999760  SS(3516)=      0.000000
SS(3517)= 199999.812000  SS(3518)=      0.000000  SS(3519)=      0.000000
SS(3520)=  24999.992200  SS(3521)=    624.999760  SS(3522)=      0.000000
SS(3523)=      0.000000  SS(3524)=      0.000000  SS(3525)=      0.000000
SS(3526)=      0.000000  SS(3527)=      0.000000  SS(3528)=      0.000000
SS(3529)=      0.000000  SS(3530)=      0.000000  SS(3531)=      0.000000
SS(3532)=  24999.992200  SS(3533)=      0.000000  SS(3534)=    624.999760
SS(3535)=   2520.832500  SS(3536)=      0.000000  SS(3537)=      0.000000
SS(3538)=  -1249.999760  SS(3539)=      0.000000  SS(3540)=      0.000000
SS(3541)=      0.000000  SS(3542)=      0.000000  SS(3543)=      0.000000
SS(3544)=      0.000000  SS(3545)=      0.000000  SS(3546)=      0.000000
SS(3547)=      0.000000  SS(3548)=   -624.999760  SS(3549)=      0.000000
SS(3550)=    -10.416666  SS(3551)=   2520.832500  SS(3552)=      0.000000
SS(3553)=   -624.999760  SS(3554)=    -10.416666  SS(3555)=      0.000000
SS(3556)=      0.000000  SS(3557)=      0.000000  SS(3558)=      0.000000
SS(3559)=      0.000000  SS(3560)=      0.000000  SS(3561)=      0.000000
SS(3562)=      0.000000  SS(3563)=      0.000000  SS(3564)=      0.000000
SS(3565)=      0.000000  SS(3566)=  -1249.999760  SS(3567)=      0.000000
SS(3568)= 199999.812000  SS(3569)=      0.000000  SS(3570)=      0.000000
SS(3571)=  24999.992200  SS(3572)=    624.999760  SS(3573)=      0.000000
SS(3574)=      0.000000  SS(3575)=      0.000000  SS(3576)=      0.000000
SS(3577)=      0.000000  SS(3578)=      0.000000  SS(3579)=      0.000000
SS(3580)=      0.000000  SS(3581)=      0.000000  SS(3582)=      0.000000
SS(3583)=  24999.992200  SS(3584)=      0.000000  SS(3585)=    624.999760
SS(3586)=   1270.832760  SS(3587)=      0.000000  SS(3588)=      0.000000
SS(3589)=  -1249.999760  SS(3590)=      0.000000  SS(3591)=      0.000000
SS(3592)=      0.000000  SS(3593)=      0.000000  SS(3594)=      0.000000
SS(3595)=      0.000000  SS(3596)=      0.000000  SS(3597)=      0.000000
SS(3598)=      0.000000  SS(3599)=   -624.999760  SS(3600)=      0.000000
SS(3601)=    -10.416666  SS(3602)=   2510.416000  SS(3603)=      0.000000
SS(3604)=   -624.999760  SS(3605)=    -10.416666  SS(3606)=      0.000000
SS(3607)=      0.000000  SS(3608)=      0.000000  SS(3609)=      0.000000
SS(3610)=      0.000000  SS(3611)=      0.000000  SS(3612)=      0.000000
SS(3613)=      0.000000  SS(3614)=      0.000000  SS(3615)=      0.000000
SS(3616)=      0.000000  SS(3617)=  -1249.999760  SS(3618)=      0.000000
SS(3619)= 149999.875000  SS(3620)=   -624.999760  SS(3621)=      0.000000
SS(3622)=  24999.992200  SS(3623)=    624.999760  SS(3624)=      0.000000
SS(3625)=      0.000000  SS(3626)=      0.000000  SS(3627)=      0.000000
SS(3628)=      0.000000  SS(3629)=      0.000000  SS(3630)=      0.000000
SS(3631)=      0.000000  SS(3632)=      0.000000  SS(3633)=      0.000000
SS(3634)=  24999.992200  SS(3635)=      0.000000  SS(3636)=    624.999760
SS(3637)=   1270.832760  SS(3638)=      0.000000  SS(3639)=      0.000000
SS(3640)=      0.000000  SS(3641)=      0.000000  SS(3642)=      0.000000
SS(3643)=      0.000000  SS(3644)=      0.000000  SS(3645)=      0.000000
SS(3646)=      0.000000  SS(3647)=      0.000000  SS(3648)=      0.000000
SS(3649)=      0.000000  SS(3650)=   -624.999760  SS(3651)=      0.000000
SS(3652)=    -10.416666  SS(3653)=   2510.416000  SS(3654)=      0.000000
SS(3655)=      0.000000  SS(3656)=      0.000000  SS(3657)=      0.000000
SS(3658)=      0.000000  SS(3659)=      0.000000  SS(3660)=      0.000000
SS(3661)=      0.000000  SS(3662)=      0.000000  SS(3663)=      0.000000
SS(3664)=      0.000000  SS(3665)=      0.000000  SS(3666)=      0.000000
SS(3667)=      0.000000  SS(3668)=  -1249.999760  SS(3669)=      0.000000
SS(3670)= 149999.875000  SS(3671)=    624.999760  SS(3672)=      0.000000
SS(3673)=      0.000000  SS(3674)=      0.000000  SS(3675)=      0.000000
SS(3676)=      0.000000  SS(3677)=      0.000000  SS(3678)=      0.000000
SS(3679)=      0.000000  SS(3680)=      0.000000  SS(3681)=      0.000000
SS(3682)=      0.000000  SS(3683)=      0.000000  SS(3684)=      0.000000
SS(3685)=  24999.992200  SS(3686)=      0.000000  SS(3687)=    624.999760
SS(3688)=   2520.832500  SS(3689)=      0.000000  SS(3690)=      0.000000
SS(3691)=  -1249.999760  SS(3692)=      0.000000  SS(3693)=      0.000000
SS(3694)=      0.000000  SS(3695)=      0.000000  SS(3696)=      0.000000
SS(3697)=      0.000000  SS(3698)=      0.000000  SS(3699)=      0.000000
SS(3700)=      0.000000  SS(3701)=   -624.999760  SS(3702)=      0.000000
SS(3703)=    -10.416666  SS(3704)=   2520.832500  SS(3705)=      0.000000
SS(3706)=   -624.999760  SS(3707)=    -10.416666  SS(3708)=      0.000000
SS(3709)=      0.000000  SS(3710)=      0.000000  SS(3711)=      0.000000
SS(3712)=      0.000000  SS(3713)=      0.000000  SS(3714)=      0.000000
SS(3715)=      0.000000  SS(3716)=      0.000000  SS(3717)=      0.000000
SS(3718)=      0.000000  SS(3719)=  -1249.999760  SS(3720)=      0.000000
SS(3721)= 199999.812000  SS(3722)=      0.000000  SS(3723)=      0.000000
SS(3724)=  24999.992200  SS(3725)=    624.999760  SS(3726)=      0.000000
SS(3727)=      0.000000  SS(3728)=      0.000000  SS(3729)=      0.000000
SS(3730)=      0.000000  SS(3731)=      0.000000  SS(3732)=      0.000000
SS(3733)=      0.000000  SS(3734)=      0.000000  SS(3735)=      0.000000
SS(3736)=  24999.992200  SS(3737)=      0.000000  SS(3738)=    624.999760
SS(3739)=   2520.832500  SS(3740)=      0.000000  SS(3741)=      0.000000
SS(3742)=  -1249.999760  SS(3743)=      0.000000  SS(3744)=      0.000000
SS(3745)=      0.000000  SS(3746)=      0.000000  SS(3747)=      0.000000
SS(3748)=      0.000000  SS(3749)=      0.000000  SS(3750)=      0.000000
SS(3751)=      0.000000  SS(3752)=   -624.999760  SS(3753)=      0.000000
SS(3754)=    -10.416666  SS(3755)=   2520.832500  SS(3756)=      0.000000
SS(3757)=   -624.999760  SS(3758)=    -10.416666  SS(3759)=      0.000000
SS(3760)=      0.000000  SS(3761)=      0.000000  SS(3762)=      0.000000
SS(3763)=      0.000000  SS(3764)=      0.000000  SS(3765)=      0.000000
SS(3766)=      0.000000  SS(3767)=      0.000000  SS(3768)=      0.000000
SS(3769)=      0.000000  SS(3770)=  -1249.999760  SS(3771)=      0.000000
SS(3772)= 199999.812000  SS(3773)=      0.000000  SS(3774)=      0.000000
SS(3775)=  24999.992200  SS(3776)=    624.999760  SS(3777)=      0.000000
SS(3778)=      0.000000  SS(3779)=      0.000000  SS(3780)=      0.000000
SS(3781)=      0.000000  SS(3782)=      0.000000  SS(3783)=      0.000000
SS(3784)=      0.000000  SS(3785)=      0.000000  SS(3786)=      0.000000
SS(3787)=  24999.992200  SS(3788)=      0.000000  SS(3789)=    624.999760
SS(3790)=   2520.832500  SS(3791)=      0.000000  SS(3792)=      0.000000
SS(3793)=  -1249.999760  SS(3794)=      0.000000  SS(3795)=      0.000000
SS(3796)=      0.000000  SS(3797)=      0.000000  SS(3798)=      0.000000
SS(3799)=      0.000000  SS(3800)=      0.000000  SS(3801)=      0.000000
SS(3802)=      0.000000  SS(3803)=   -624.999760  SS(3804)=      0.000000
SS(3805)=    -10.416666  SS(3806)=   2520.832500  SS(3807)=      0.000000
SS(3808)=   -624.999760  SS(3809)=    -10.416666  SS(3810)=      0.000000
SS(3811)=      0.000000  SS(3812)=      0.000000  SS(3813)=      0.000000
SS(3814)=      0.000000  SS(3815)=      0.000000  SS(3816)=      0.000000
SS(3817)=      0.000000  SS(3818)=      0.000000  SS(3819)=      0.000000
SS(3820)=      0.000000  SS(3821)=  -1249.999760  SS(3822)=      0.000000
SS(3823)= 199999.812000  SS(3824)=      0.000000  SS(3825)=      0.000000
SS(3826)=  24999.992200  SS(3827)=    624.999760  SS(3828)=      0.000000
SS(3829)=      0.000000  SS(3830)=      0.000000  SS(3831)=      0.000000
SS(3832)=      0.000000  SS(3833)=      0.000000  SS(3834)=      0.000000
SS(3835)=      0.000000  SS(3836)=      0.000000  SS(3837)=      0.000000
SS(3838)=  24999.992200  SS(3839)=      0.000000  SS(3840)=    624.999760
SS(3841)=   1270.832760  SS(3842)=      0.000000  SS(3843)=      0.000000
SS(3844)=  -1249.999760  SS(3845)=      0.000000  SS(3846)=      0.000000
SS(3847)=      0.000000  SS(3848)=      0.000000  SS(3849)=      0.000000
SS(3850)=      0.000000  SS(3851)=      0.000000  SS(3852)=      0.000000
SS(3853)=      0.000000  SS(3854)=   -624.999760  SS(3855)=      0.000000
SS(3856)=    -10.416666  SS(3857)=   2510.416000  SS(3858)=      0.000000
SS(3859)=   -624.999760  SS(3860)=    -10.416666  SS(3861)=      0.000000
SS(3862)=      0.000000  SS(3863)=      0.000000  SS(3864)=      0.000000
SS(3865)=      0.000000  SS(3866)=      0.000000  SS(3867)=      0.000000
SS(3868)=      0.000000  SS(3869)=      0.000000  SS(3870)=      0.000000
SS(3871)=      0.000000  SS(3872)=  -1249.999760  SS(3873)=      0.000000
SS(3874)= 149999.875000  SS(3875)=   -624.999760  SS(3876)=      0.000000
SS(3877)=  24999.992200  SS(3878)=    624.999760  SS(3879)=      0.000000
SS(3880)=      0.000000  SS(3881)=      0.000000  SS(3882)=      0.000000
SS(3883)=      0.000000  SS(3884)=      0.000000  SS(3885)=      0.000000
SS(3886)=        0.000000  SS(3887)=        0.000000  SS(3888)=        0.000000
SS(3889)=   24999.992200  SS(3890)=        0.000000  SS(3891)=      624.999760

  LOAD CONDITION  1

FORCES
Q(  1)=         0.000000  Q(  2)=       -11.999992  Q(  3)=      -239.999832
Q(  4)=         0.000000  Q(  5)=       -23.999985  Q(  6)=         0.000000
Q(  7)=         0.000000  Q(  8)=       -23.999985  Q(  9)=         0.000000
Q( 10)=         0.000000  Q( 11)=       -23.999985  Q( 12)=         0.000000
Q( 13)=         0.000000  Q( 14)=       -11.999992  Q( 15)=       239.999832
Q( 16)=         0.000000  Q( 17)=       -11.999992  Q( 18)=      -239.999832
Q( 19)=         0.000000  Q( 20)=       -23.999985  Q( 21)=         0.000000
Q( 22)=         0.000000  Q( 23)=       -23.999985  Q( 24)=         0.000000
Q( 25)=         0.000000  Q( 26)=       -23.999985  Q( 27)=         0.000000
Q( 28)=         0.000000  Q( 29)=       -11.999992  Q( 30)=       239.999832
Q( 31)=         0.000000  Q( 32)=       -11.999992  Q( 33)=      -239.999832
Q( 34)=         0.000000  Q( 35)=       -23.999985  Q( 36)=         0.000000
Q( 37)=         0.000000  Q( 38)=       -23.999985  Q( 39)=         0.000000
Q( 40)=         0.000000  Q( 41)=       -23.999985  Q( 42)=         0.000000
Q( 43)=         0.000000  Q( 44)=       -11.999992  Q( 45)=       239.999832
Q( 46)=         0.000000  Q( 47)=       -11.999992  Q( 48)=      -239.999832
Q( 49)=         0.000000  Q( 50)=       -23.999985  Q( 51)=         0.000000
Q( 52)=         0.000000  Q( 53)=       -23.999985  Q( 54)=         0.000000
Q( 55)=         0.000000  Q( 56)=       -23.999985  Q( 57)=         0.000000
Q( 58)=         0.000000  Q( 59)=       -11.999992  Q( 60)=       239.999832
Q( 61)=         0.000000  Q( 62)=       -11.999992  Q( 63)=      -239.999832
Q( 64)=         0.000000  Q( 65)=       -23.999985  Q( 66)=         0.000000
Q( 67)=         0.000000  Q( 68)=       -23.999985  Q( 69)=         0.000000
Q( 70)=         0.000000  Q( 71)=       -23.999985  Q( 72)=         0.000000
Q( 73)=         0.000000  Q( 74)=       -11.999992  Q( 75)=       239.999832
Q( 76)=         0.000000  Q( 77)=       -11.999992  Q( 78)=      -239.999832
Q( 79)=         0.000000  Q( 80)=       -23.999985  Q( 81)=         0.000000
Q( 82)=         0.000000  Q( 83)=       -23.999985  Q( 84)=         0.000000
Q( 85)=         0.000000  Q( 86)=       -23.999985  Q( 87)=         0.000000
Q( 88)=         0.000000  Q( 89)=       -11.999992  Q( 90)=       239.999832
Q( 91)=         0.000000  Q( 92)=       -11.999992  Q( 93)=      -239.999832
Q( 94)=         0.000000  Q( 95)=       -23.999985  Q( 96)=         0.000000
Q( 97)=         0.000000  Q( 98)=       -23.999985  Q( 99)=         0.000000
Q(100)=         0.000000  Q(101)=       -23.999985  Q(102)=         0.000000
Q(103)=         0.000000  Q(104)=       -11.999992  Q(105)=       239.999832
Q(106)=         0.000000  Q(107)=       -11.999992  Q(108)=      -239.999832
Q(109)=         0.000000  Q(110)=       -23.999985  Q(111)=         0.000000
Q(112)=         0.000000  Q(113)=       -23.999985  Q(114)=         0.000000
Q(115)=         0.000000  Q(116)=       -23.999985  Q(117)=         0.000000
Q(118)=         0.000000  Q(119)=       -11.999992  Q(120)=       239.999832
Q(121)=         0.000000  Q(122)=       -11.999992  Q(123)=      -239.999832
Q(124)=         0.000000  Q(125)=       -23.999985  Q(126)=         0.000000
Q(127)=         0.000000  Q(128)=       -23.999985  Q(129)=         0.000000
Q(130)=         0.000000  Q(131)=       -23.999985  Q(132)=         0.000000
Q(133)=         0.000000  Q(134)=       -11.999992  Q(135)=       239.999832
Q(136)=         0.000000  Q(137)=       -11.999992  Q(138)=      -239.999832
Q(139)=         0.000000  Q(140)=       -23.999985  Q(141)=         0.000000
Q(142)=         0.000000  Q(143)=       -23.999985  Q(144)=         0.000000
Q(145)=         0.000000  Q(146)=       -23.999985  Q(147)=         0.000000
Q(148)=         0.000000  Q(149)=       -11.999992  Q(150)=       239.999832
Q(151)=         0.000000  Q(152)=       -11.999992  Q(153)=      -239.999832
Q(154)=         0.000000  Q(155)=       -23.999985  Q(156)=         0.000000
Q(157)=         0.000000  Q(158)=       -23.999985  Q(159)=         0.000000
Q(160)=         0.000000  Q(161)=       -23.999985  Q(162)=         0.000000
Q(163)=         0.000000  Q(164)=       -11.999992  Q(165)=       239.999832
```

```
Q(166)=        0.000000  Q(167)=     -11.999992  Q(168)=     -239.999832
Q(169)=        0.000000  Q(170)=     -23.999985  Q(171)=        0.000000
Q(172)=        0.000000  Q(173)=     -23.999985  Q(174)=        0.000000
Q(175)=        0.000000  Q(176)=     -23.999985  Q(177)=        0.000000
Q(178)=        0.000000  Q(179)=     -11.999992  Q(180)=      239.999832
Q(181)=        0.000000  Q(182)=     -11.999992  Q(183)=     -239.999832
Q(184)=        0.000000  Q(185)=     -23.999985  Q(186)=        0.000000
Q(187)=        0.000000  Q(188)=     -23.999985  Q(189)=        0.000000
Q(190)=        0.000000  Q(191)=     -23.999985  Q(192)=        0.000000
Q(193)=        0.000000  Q(194)=     -11.999992  Q(195)=      239.999832
Q(196)=        0.000000  Q(197)=     -11.999992  Q(198)=     -239.999832
Q(199)=        0.000000  Q(200)=     -23.999985  Q(201)=        0.000000
Q(202)=        0.000000  Q(203)=     -23.999985  Q(204)=        0.000000
Q(205)=        0.000000  Q(206)=     -23.999985  Q(207)=        0.000000
Q(208)=        0.000000  Q(209)=     -11.999992  Q(210)=      239.999832
Q(211)=        0.000000  Q(212)=     -11.999992  Q(213)=     -239.999832
Q(214)=        0.000000  Q(215)=     -23.999985  Q(216)=        0.000000
Q(217)=        0.000000  Q(218)=     -23.999985  Q(219)=        0.000000
Q(220)=        0.000000  Q(221)=     -23.999985  Q(222)=        0.000000
Q(223)=        0.000000  Q(224)=     -11.999992  Q(225)=      239.999832
Q(226)=        0.000000  Q(227)=     -11.999992  Q(228)=     -239.999832
Q(229)=        0.000000  Q(230)=     -23.999985  Q(231)=        0.000000
Q(232)=        0.000000  Q(233)=     -23.999985  Q(234)=        0.000000
Q(235)=        0.000000  Q(236)=     -23.999985  Q(237)=        0.000000
Q(238)=        0.000000  Q(239)=     -11.999992  Q(240)=      239.999832

    START 14:11:45
    FINISH 14:12:41
    Time elapsed in Solution Process: 55.60156 Seconds


DISPLACEMENTS
Q(  1)=        0.010027  Q(  2)=      -1.644572  Q(  3)=       -0.005464
Q(  4)=        0.006034  Q(  5)=      -2.327070  Q(  6)=       -0.002389
Q(  7)=        0.000002  Q(  8)=      -2.501757  Q(  9)=       -0.000000
Q( 10)=       -0.006030  Q( 11)=      -2.327071  Q( 12)=        0.002389
Q( 13)=       -0.010022  Q( 14)=      -1.644573  Q( 15)=        0.005464
Q( 16)=       -0.002333  Q( 17)=      -1.633211  Q( 18)=       -0.002726
Q( 19)=       -0.001282  Q( 20)=      -2.309370  Q( 21)=       -0.001813
Q( 22)=        0.000002  Q( 23)=      -2.483079  Q( 24)=       -0.000000
Q( 25)=        0.001286  Q( 26)=      -2.309370  Q( 27)=        0.001813
Q( 28)=        0.002336  Q( 29)=      -1.633212  Q( 30)=        0.002726
Q( 31)=        0.000197  Q( 32)=      -1.608885  Q( 33)=       -0.003114
Q( 34)=        0.000078  Q( 35)=      -2.275292  Q( 36)=       -0.001778
Q( 37)=        0.000002  Q( 38)=      -2.446283  Q( 39)=       -0.000000
Q( 40)=       -0.000074  Q( 41)=      -2.275293  Q( 42)=        0.001778
Q( 43)=       -0.000194  Q( 44)=      -1.608886  Q( 45)=        0.003114
Q( 46)=       -0.000172  Q( 47)=      -1.571851  Q( 48)=       -0.003014
Q( 49)=       -0.000106  Q( 50)=      -2.224586  Q( 51)=       -0.001751
Q( 52)=        0.000001  Q( 53)=      -2.391357  Q( 54)=       -0.000000
Q( 55)=        0.000108  Q( 56)=      -2.224587  Q( 57)=        0.001751
Q( 58)=        0.000175  Q( 59)=      -1.571852  Q( 60)=        0.003014
Q( 61)=       -0.000161  Q( 62)=      -1.522161  Q( 63)=       -0.002976
Q( 64)=       -0.000111  Q( 65)=      -2.157222  Q( 66)=       -0.001694
Q( 67)=        0.000001  Q( 68)=      -2.318259  Q( 69)=       -0.000000
Q( 70)=        0.000113  Q( 71)=      -2.157222  Q( 72)=        0.001694
Q( 73)=        0.000164  Q( 74)=      -1.522161  Q( 75)=        0.002976
Q( 76)=       -0.000205  Q( 77)=      -1.459912  Q( 78)=       -0.002914
Q( 79)=       -0.000138  Q( 80)=      -2.073119  Q( 81)=       -0.001628
Q( 82)=        0.000001  Q( 83)=      -2.226951  Q( 84)=       -0.000000
Q( 85)=        0.000141  Q( 86)=      -2.073120  Q( 87)=        0.001628
Q( 88)=        0.000207  Q( 89)=      -1.459913  Q( 90)=        0.002914
Q( 91)=       -0.000242  Q( 92)=      -1.385225  Q( 93)=       -0.002843
```

```
Q( 94)=      -0.000164  Q( 95)=      -1.972187  Q( 96)=      -0.001548
Q( 97)=       0.000001  Q( 98)=      -2.117379  Q( 99)=      -0.000000
Q(100)=       0.000166  Q(101)=      -1.972187  Q(102)=       0.001548
Q(103)=       0.000244  Q(104)=      -1.385225  Q(105)=       0.002843
Q(106)=      -0.000283  Q(107)=      -1.298241  Q(108)=      -0.002758
Q(109)=      -0.000192  Q(110)=      -1.854314  Q(111)=      -0.001454
Q(112)=       0.000001  Q(113)=      -1.989478  Q(114)=      -0.000000
Q(115)=       0.000193  Q(116)=      -1.854314  Q(117)=       0.001454
Q(118)=       0.000285  Q(119)=      -1.298241  Q(120)=       0.002758
Q(121)=      -0.000326  Q(122)=      -1.199129  Q(123)=      -0.002660
Q(124)=      -0.000220  Q(125)=      -1.719370  Q(126)=      -0.001346
Q(127)=       0.000001  Q(128)=      -1.843176  Q(129)=      -0.000000
Q(130)=       0.000222  Q(131)=      -1.719370  Q(132)=       0.001346
Q(133)=       0.000328  Q(134)=      -1.199129  Q(135)=       0.002660
Q(136)=      -0.000372  Q(137)=      -1.088084  Q(138)=      -0.002547
Q(139)=      -0.000251  Q(140)=      -1.567198  Q(141)=      -0.001223
Q(142)=       0.000001  Q(143)=      -1.678391  Q(144)=      -0.000000
Q(145)=       0.000252  Q(146)=      -1.567199  Q(147)=       0.001223
Q(148)=       0.000373  Q(149)=      -1.088084  Q(150)=       0.002547
Q(151)=      -0.000421  Q(152)=      -0.965331  Q(153)=      -0.002418
Q(154)=      -0.000283  Q(155)=      -1.397619  Q(156)=      -0.001084
Q(157)=       0.000001  Q(158)=      -1.495036  Q(159)=      -0.000000
Q(160)=       0.000285  Q(161)=      -1.397620  Q(162)=       0.001084
Q(163)=       0.000422  Q(164)=      -0.965331  Q(165)=       0.002418
Q(166)=      -0.000474  Q(167)=      -0.831126  Q(168)=      -0.002272
Q(169)=      -0.000318  Q(170)=      -1.210421  Q(171)=      -0.000929
Q(172)=       0.000001  Q(173)=      -1.293020  Q(174)=      -0.000000
Q(175)=       0.000320  Q(176)=      -1.210422  Q(177)=       0.000929
Q(178)=       0.000475  Q(179)=      -0.831126  Q(180)=       0.002272
Q(181)=      -0.000527  Q(182)=      -0.685761  Q(183)=      -0.002106
Q(184)=      -0.000354  Q(185)=      -1.005360  Q(186)=      -0.000758
Q(187)=       0.000000  Q(188)=      -1.072251  Q(189)=      -0.000000
Q(190)=       0.000355  Q(191)=      -1.005360  Q(192)=       0.000758
Q(193)=       0.000528  Q(194)=      -0.685761  Q(195)=       0.002106
Q(196)=      -0.000616  Q(197)=      -0.529565  Q(198)=      -0.001927
Q(199)=      -0.000404  Q(200)=      -0.782151  Q(201)=      -0.000565
Q(202)=       0.000000  Q(203)=      -0.832638  Q(204)=      -0.000000
Q(205)=       0.000405  Q(206)=      -0.782151  Q(207)=       0.000565
Q(208)=       0.000617  Q(209)=      -0.529565  Q(210)=       0.001927
Q(211)=      -0.000501  Q(212)=      -0.362910  Q(213)=      -0.001679
Q(214)=      -0.000373  Q(215)=      -0.540463  Q(216)=      -0.000371
Q(217)=       0.000000  Q(218)=      -0.574101  Q(219)=      -0.000000
Q(220)=       0.000374  Q(221)=      -0.540463  Q(222)=       0.000371
Q(223)=       0.000502  Q(224)=      -0.362910  Q(225)=       0.001679
Q(226)=      -0.001800  Q(227)=      -0.186201  Q(228)=      -0.001694
Q(229)=      -0.000986  Q(230)=      -0.279934  Q(231)=      -0.000086
Q(232)=       0.000000  Q(233)=      -0.296555  Q(234)=      -0.000000
Q(235)=       0.000986  Q(236)=      -0.279934  Q(237)=       0.000086
Q(238)=       0.001800  Q(239)=      -0.186201  Q(240)=       0.001694

  LOAD CONDITION  2

FORCES
Q(  1)=       1.500000  Q(  2)=       0.000000  Q(  3)=       0.000000
Q(  4)=       1.500000  Q(  5)=       0.000000  Q(  6)=       0.000000
Q(  7)=       1.500000  Q(  8)=       0.000000  Q(  9)=       0.000000
Q( 10)=       1.500000  Q( 11)=       0.000000  Q( 12)=       0.000000
Q( 13)=       1.500000  Q( 14)=       0.000000  Q( 15)=       0.000000
Q( 16)=       1.500000  Q( 17)=       0.000000  Q( 18)=       0.000000
Q( 19)=       1.500000  Q( 20)=       0.000000  Q( 21)=       0.000000
Q( 22)=       1.500000  Q( 23)=       0.000000  Q( 24)=       0.000000
```

```
Q( 25)=      1.500000  Q( 26)=      0.000000  Q( 27)=      0.000000
Q( 28)=      1.500000  Q( 29)=      0.000000  Q( 30)=      0.000000
Q( 31)=      1.500000  Q( 32)=      0.000000  Q( 33)=      0.000000
Q( 34)=      1.500000  Q( 35)=      0.000000  Q( 36)=      0.000000
Q( 37)=      1.500000  Q( 38)=      0.000000  Q( 39)=      0.000000
Q( 40)=      1.500000  Q( 41)=      0.000000  Q( 42)=      0.000000
Q( 43)=      1.500000  Q( 44)=      0.000000  Q( 45)=      0.000000
Q( 46)=      1.500000  Q( 47)=      0.000000  Q( 48)=      0.000000
Q( 49)=      1.500000  Q( 50)=      0.000000  Q( 51)=      0.000000
Q( 52)=      1.500000  Q( 53)=      0.000000  Q( 54)=      0.000000
Q( 55)=      1.500000  Q( 56)=      0.000000  Q( 57)=      0.000000
Q( 58)=      1.500000  Q( 59)=      0.000000  Q( 60)=      0.000000
Q( 61)=      1.500000  Q( 62)=      0.000000  Q( 63)=      0.000000
Q( 64)=      1.500000  Q( 65)=      0.000000  Q( 66)=      0.000000
Q( 67)=      1.500000  Q( 68)=      0.000000  Q( 69)=      0.000000
Q( 70)=      1.500000  Q( 71)=      0.000000  Q( 72)=      0.000000
Q( 73)=      1.500000  Q( 74)=      0.000000  Q( 75)=      0.000000
Q( 76)=      1.500000  Q( 77)=      0.000000  Q( 78)=      0.000000
Q( 79)=      1.500000  Q( 80)=      0.000000  Q( 81)=      0.000000
Q( 82)=      1.500000  Q( 83)=      0.000000  Q( 84)=      0.000000
Q( 85)=      1.500000  Q( 86)=      0.000000  Q( 87)=      0.000000
Q( 88)=      1.500000  Q( 89)=      0.000000  Q( 90)=      0.000000
Q( 91)=      1.500000  Q( 92)=      0.000000  Q( 93)=      0.000000
Q( 94)=      1.500000  Q( 95)=      0.000000  Q( 96)=      0.000000
Q( 97)=      1.500000  Q( 98)=      0.000000  Q( 99)=      0.000000
Q(100)=      1.500000  Q(101)=      0.000000  Q(102)=      0.000000
Q(103)=      1.500000  Q(104)=      0.000000  Q(105)=      0.000000
Q(106)=      1.500000  Q(107)=      0.000000  Q(108)=      0.000000
Q(109)=      1.500000  Q(110)=      0.000000  Q(111)=      0.000000
Q(112)=      1.500000  Q(113)=      0.000000  Q(114)=      0.000000
Q(115)=      1.500000  Q(116)=      0.000000  Q(117)=      0.000000
Q(118)=      1.500000  Q(119)=      0.000000  Q(120)=      0.000000
Q(121)=      1.500000  Q(122)=      0.000000  Q(123)=      0.000000
Q(124)=      1.500000  Q(125)=      0.000000  Q(126)=      0.000000
Q(127)=      1.500000  Q(128)=      0.000000  Q(129)=      0.000000
Q(130)=      1.500000  Q(131)=      0.000000  Q(132)=      0.000000
Q(133)=      1.500000  Q(134)=      0.000000  Q(135)=      0.000000
Q(136)=      1.500000  Q(137)=      0.000000  Q(138)=      0.000000
Q(139)=      1.500000  Q(140)=      0.000000  Q(141)=      0.000000
Q(142)=      1.500000  Q(143)=      0.000000  Q(144)=      0.000000
Q(145)=      1.500000  Q(146)=      0.000000  Q(147)=      0.000000
Q(148)=      1.500000  Q(149)=      0.000000  Q(150)=      0.000000
Q(151)=      1.500000  Q(152)=      0.000000  Q(153)=      0.000000
Q(154)=      1.500000  Q(155)=      0.000000  Q(156)=      0.000000
Q(157)=      1.500000  Q(158)=      0.000000  Q(159)=      0.000000
Q(160)=      1.500000  Q(161)=      0.000000  Q(162)=      0.000000
Q(163)=      1.500000  Q(164)=      0.000000  Q(165)=      0.000000
Q(166)=      1.500000  Q(167)=      0.000000  Q(168)=      0.000000
Q(169)=      1.500000  Q(170)=      0.000000  Q(171)=      0.000000
Q(172)=      1.500000  Q(173)=      0.000000  Q(174)=      0.000000
Q(175)=      1.500000  Q(176)=      0.000000  Q(177)=      0.000000
Q(178)=      1.500000  Q(179)=      0.000000  Q(180)=      0.000000
Q(181)=      1.500000  Q(182)=      0.000000  Q(183)=      0.000000
Q(184)=      1.500000  Q(185)=      0.000000  Q(186)=      0.000000
Q(187)=      1.500000  Q(188)=      0.000000  Q(189)=      0.000000
Q(190)=      1.500000  Q(191)=      0.000000  Q(192)=      0.000000
Q(193)=      1.500000  Q(194)=      0.000000  Q(195)=      0.000000
Q(196)=      1.500000  Q(197)=      0.000000  Q(198)=      0.000000
Q(199)=      1.500000  Q(200)=      0.000000  Q(201)=      0.000000
Q(202)=      1.500000  Q(203)=      0.000000  Q(204)=      0.000000
Q(205)=      1.500000  Q(206)=      0.000000  Q(207)=      0.000000
Q(208)=      1.500000  Q(209)=      0.000000  Q(210)=      0.000000
Q(211)=      1.500000  Q(212)=      0.000000  Q(213)=      0.000000
```

```
Q(214)=      1.500000  Q(215)=        0.000000  Q(216)=        0.000000
Q(217)=      1.500000  Q(218)=        0.000000  Q(219)=        0.000000
Q(220)=      1.500000  Q(221)=        0.000000  Q(222)=        0.000000
Q(223)=      1.500000  Q(224)=        0.000000  Q(225)=        0.000000
Q(226)=      1.500000  Q(227)=        0.000000  Q(228)=        0.000000
Q(229)=      1.500000  Q(230)=        0.000000  Q(231)=        0.000000
Q(232)=      1.500000  Q(233)=        0.000000  Q(234)=        0.000000
Q(235)=      1.500000  Q(236)=        0.000000  Q(237)=        0.000000
Q(238)=      1.500000  Q(239)=        0.000000  Q(240)=        0.000000

     START 14:12:46
     FINISH 14:12:57
     Time elapsed in Solution Process: 10.10156 Seconds


DISPLACEMENTS
Q(  1)=     50.395107  Q(  2)=        1.007840  Q(  3)=       -0.006996
Q(  4)=     50.393776  Q(  5)=        0.247385  Q(  6)=       -0.005108
Q(  7)=     50.392811  Q(  8)=        0.000000  Q(  9)=       -0.003151
Q( 10)=     50.393780  Q( 11)=       -0.247385  Q( 12)=       -0.005108
Q( 13)=     50.395119  Q( 14)=       -1.007841  Q( 15)=       -0.006996
Q( 16)=     49.513630  Q( 17)=        1.008125  Q( 18)=       -0.007955
Q( 19)=     49.513569  Q( 20)=        0.245032  Q( 21)=       -0.006405
Q( 22)=     49.513660  Q( 23)=       -0.000000  Q( 24)=       -0.005228
Q( 25)=     49.513557  Q( 26)=       -0.245032  Q( 27)=       -0.006405
Q( 28)=     49.513611  Q( 29)=       -1.008125  Q( 30)=       -0.007955
Q( 31)=     48.312286  Q( 32)=        1.007588  Q( 33)=       -0.010014
Q( 34)=     48.311890  Q( 35)=        0.239726  Q( 36)=       -0.007716
Q( 37)=     48.311760  Q( 38)=       -0.000000  Q( 39)=       -0.006495
Q( 40)=     48.311897  Q( 41)=       -0.239726  Q( 42)=       -0.007716
Q( 43)=     48.312294  Q( 44)=       -1.007589  Q( 45)=       -0.010014
Q( 46)=     46.773376  Q( 47)=        1.004586  Q( 48)=       -0.011949
Q( 49)=     46.773022  Q( 50)=        0.231778  Q( 51)=       -0.009064
Q( 52)=     46.772911  Q( 53)=       -0.000000  Q( 54)=       -0.007883
Q( 55)=     46.773010  Q( 56)=       -0.231778  Q( 57)=       -0.009064
Q( 58)=     46.773357  Q( 59)=       -1.004586  Q( 60)=       -0.011949
Q( 61)=     44.898815  Q( 62)=        0.997516  Q( 63)=       -0.013886
Q( 64)=     44.898457  Q( 65)=        0.221354  Q( 66)=       -0.010388
Q( 67)=     44.898346  Q( 68)=       -0.000000  Q( 69)=       -0.009235
Q( 70)=     44.898453  Q( 71)=       -0.221354  Q( 72)=       -0.010388
Q( 73)=     44.898808  Q( 74)=       -0.997516  Q( 75)=       -0.013886
Q( 76)=     42.691734  Q( 77)=        0.984778  Q( 78)=       -0.015791
Q( 79)=     42.691376  Q( 80)=        0.208632  Q( 81)=       -0.011687
Q( 82)=     42.691257  Q( 83)=       -0.000000  Q( 84)=       -0.010565
Q( 85)=     42.691364  Q( 86)=       -0.208632  Q( 87)=       -0.011687
Q( 88)=     42.691711  Q( 89)=       -0.984777  Q( 90)=       -0.015791
Q( 91)=     40.155945  Q( 92)=        0.964768  Q( 93)=       -0.017659
Q( 94)=     40.155594  Q( 95)=        0.193793  Q( 96)=       -0.012953
Q( 97)=     40.155487  Q( 98)=       -0.000000  Q( 99)=       -0.011871
Q(100)=     40.155582  Q(101)=       -0.193793  Q(102)=       -0.012953
Q(103)=     40.155926  Q(104)=       -0.964768  Q(105)=       -0.017659
Q(106)=     37.295593  Q(107)=        0.935877  Q(108)=       -0.019482
Q(109)=     37.295250  Q(110)=        0.177038  Q(111)=       -0.014181
Q(112)=     37.295151  Q(113)=       -0.000000  Q(114)=       -0.013152
Q(115)=     37.295242  Q(116)=       -0.177038  Q(117)=       -0.014181
Q(118)=     37.295574  Q(119)=       -0.935877  Q(120)=       -0.019482
Q(121)=     34.115707  Q(122)=        0.896478  Q(123)=       -0.021247
Q(124)=     34.115379  Q(125)=        0.158600  Q(126)=       -0.015364
Q(127)=     34.115295  Q(128)=       -0.000000  Q(129)=       -0.014406
Q(130)=     34.115376  Q(131)=       -0.158600  Q(132)=       -0.015364
Q(133)=     34.115700  Q(134)=       -0.896478  Q(135)=       -0.021247
Q(136)=     30.622490  Q(137)=        0.844922  Q(138)=       -0.022944
Q(139)=     30.622175  Q(140)=        0.138755  Q(141)=       -0.016492
```

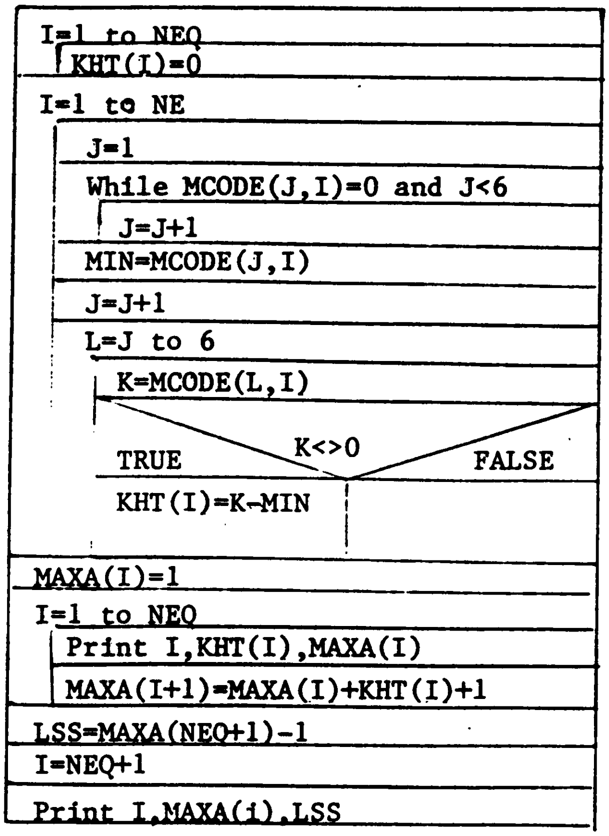
```
Q(142)=       30.622107  Q(143)=       -0.000000  Q(144)=       -0.015629
Q(145)=       30.622175  Q(146)=       -0.138755  Q(147)=       -0.016492
Q(148)=       30.622486  Q(149)=       -0.844922  Q(150)=       -0.022944
Q(151)=       26.823284  Q(152)=        0.779533  Q(153)=       -0.024560
Q(154)=       26.822990  Q(155)=        0.117839  Q(156)=       -0.017558
Q(157)=       26.822935  Q(158)=       -0.000000  Q(159)=       -0.016821
Q(160)=       26.822981  Q(161)=       -0.117839  Q(162)=       -0.017558
Q(163)=       26.823273  Q(164)=       -0.779533  Q(165)=       -0.024560
Q(166)=       22.726273  Q(167)=        0.698599  Q(168)=       -0.026085
Q(169)=       22.726004  Q(170)=        0.096260  Q(171)=       -0.018554
Q(172)=       22.725969  Q(173)=       -0.000000  Q(174)=       -0.017981
Q(175)=       22.725998  Q(176)=       -0.096260  Q(177)=       -0.018554
Q(178)=       22.726265  Q(179)=       -0.698599  Q(180)=       -0.026085
Q(181)=       18.340468  Q(182)=        0.600364  Q(183)=       -0.027502
Q(184)=       18.340216  Q(185)=        0.074516  Q(186)=       -0.019475
Q(187)=       18.340210  Q(188)=       -0.000000  Q(189)=       -0.019105
Q(190)=       18.340212  Q(191)=       -0.074516  Q(192)=       -0.019475
Q(193)=       18.340462  Q(194)=       -0.600364  Q(195)=       -0.027502
Q(196)=       13.676907  Q(197)=        0.483023  Q(198)=       -0.028806
Q(199)=       13.676740  Q(200)=        0.053210  Q(201)=       -0.020280
Q(202)=       13.676770  Q(203)=       -0.000000  Q(204)=       -0.020184
Q(205)=       13.676740  Q(206)=       -0.053210  Q(207)=       -0.020280
Q(208)=       13.676905  Q(209)=       -0.483023  Q(210)=       -0.028806
Q(211)=        8.758620  Q(212)=        0.344720  Q(213)=       -0.029672
Q(214)=        8.758329  Q(215)=        0.033076  Q(216)=       -0.020920
Q(217)=        8.758391  Q(218)=       -0.000000  Q(219)=       -0.021066
Q(220)=        8.758330  Q(221)=       -0.033076  Q(222)=       -0.020920
Q(223)=        8.758621  Q(224)=       -0.344720  Q(225)=       -0.029672
Q(226)=        3.716084  Q(227)=        0.183718  Q(228)=       -0.029053
Q(229)=        3.718665  Q(230)=        0.014924  Q(231)=       -0.019193
Q(232)=        3.719497  Q(233)=       -0.000000  Q(234)=       -0.020032
Q(235)=        3.718666  Q(236)=       -0.014924  Q(237)=       -0.019193
Q(238)=        3.716085  Q(239)=       -0.183718  Q(240)=       -0.029053
```

APPENDIX F - QUICKBASIC LISTING

```
'************************************************************************
'*                              SKYACC                                  *
'************************************************************************
'
'     PURPOSE: INPUT Q FROM A KNOWN FILE AND SOME NEW FILE.
'              THE ACCURACY OF THE NEW Q WILL BE DETERMINED
'              BY FINDING THE DIFFERENCE FROM THE CORRECT Q.
'
'     INPUT :NEQ,QCORR,QNEW
'     OUTPUT:QDIFF(*)
'
'----------------------
'  IDENTIFIERS
'----------------------
'
'      NEQ          NUMBER OF EQUATIONS                  INTEGER
'      QCORR(NEQ)   CORRECT DISPLACEMENTS                REAL
'      QNEW(NEQ)    NEW DISPLACEMENTS                    REAL
'      QDIFF(NEQ)   DIFFERENCE BETWEEN QCORR AND QNEW    REAL
'      CORRFILE     NAME OF CORRECT DATA FILE            STRING
'      NEWFILE      NAME OF NEW DATA FILE                STRING
'
'
'
'
'
       INPUT "What is the name of the Correct File";CORRFILE$
       INPUT "What is the name of the New File";NEWFILE$
       CLS
'
' OPEN DATA FILES
'
       OPEN "I",#1,CORRFILE$
       OPEN "I",#2,NEWFILE$
       OPEN "O",#3,"SKYDIFF.OUT"
'
       Q1$= "Q(###)= ######.#######  "
       Q2$=Q1$+Q1$
       Q3$=Q1$+Q1$+Q1$
       PRINT #3,:PRINT#3,
       PRINT #3,"Difference in Displacement Vectros (QCORR-QNEW)"
'
' INPUT NEQ
' DIMENSIONS ARRAYS
'
       INPUT #1,NEQ
       DIM QCORR#(NEQ),QNEW#(NEQ),QDIFF#(NEQ)
'
       NUM=INT(NEQ/3)
       FOR I=1 TO NUM
```

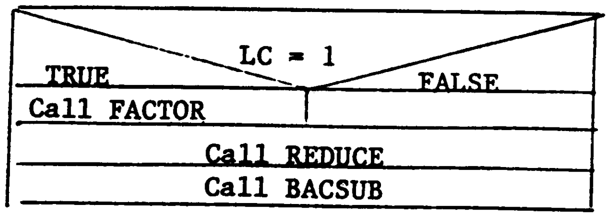
```
        INPUT #1,QCORR#(I*3-2),QCORR#(I*3-1),QCORR#(I*3)
        INPUT #2,QNEW#(I*3-2),QNEW#(I*3-1),QNEW#(I*3)
        QDIFF#(I*3-2)=QCORR#(I*3-2)-QNEW#(I*3-2)
        QDIFF#(I*3-1)=QCORR#(I*3-1)-QNEW#(I*3-1)
        QDIFF#(I*3)=QCORR#(I*3)-QNEW#(I*3)
        PRINT #3, USING Q3$;I*3-2,QDIFF#(I*3-2),I*3-1,_
             QDIFF#(I*3-1),I*3,QDIFF#(I*3)
         NEXT I
'
         NUM=NEQ-(NUM*3)
         IF (NUM <> 0) THEN
           IF (NUM = 1) THEN
          INPUT #1,QCORR#(NEQ-NUM+1)
          INPUT #2,QNEW#(NEQ-NUM+1)
          QDIFF#(NEQ-NUM+1)=QCORR#(NEQ-NUM+1)-_
                            QNEW#(NEQ-NUM+1)
          PRINT #3, USING Q1$;NEQ-NUM+1,QDIFF#(NEQ-NUM+1)
           ELSEIF (NUM = 2) THEN
          INPUT #1,QCORR#(NEQ-NUM+1),QCORR#(NEQ-NUM+2)
          INPUT #2,QNEW#(NEQ-NUM+1),QNEW#(NEQ-NUM+2)
          QDIFF#(NEQ-NUM+1)=QCORR#(NEQ-NUM+1)-_
                            QNEW#(NEQ-NUM+1)
          QDIFF#(NEQ-NUM+2)=QCORR#(NEQ-NUM+2)-_
                            QNEW#(NEQ-NUM+2)
          PRINT #3, USING Q2$;NEQ-NUM+1,QDIFF#(NEQ-NUM+1),_
                NEQ-NUM+2,QDIFF#(NEQ-NUM+2)
           END IF
         END IF
'
'DETERMINE AVERAGE DIFFERENCE
'
         FOR I=1 TO NEQ
       SUM#=SUM#+QDIFF#(I)
         NEXT I
         AVERAGE#=SUM#/NEQ
         PRINT #3,:PRINT#3,
         PRINT #3, USING "The average difference is #.#######^^^^";AVERAGE#
```

APPENDIX G - NS DIAGRAMS FOR SOLVE ROUTINES



```
I=1 to NEQ
    KHT(I)=0
I=1 to NE
    J=1
    While MCODE(J,I)=0 and J<6
        J=J+1
    MIN=MCODE(J,I)
    J=J+1
    L=J to 6
        K=MCODE(L,I)
        K<>0
        TRUE                      FALSE
        KHT(I)=K-MIN
MAXA(I)=1
I=1 to NEQ
    Print I,KHT(I),MAXA(I)
    MAXA(I+1)=MAXA(I)+KHT(I)+1
LSS=MAXA(NEQ+1)-1
I=NEQ+1
Print I,MAXA(i),LSS
```

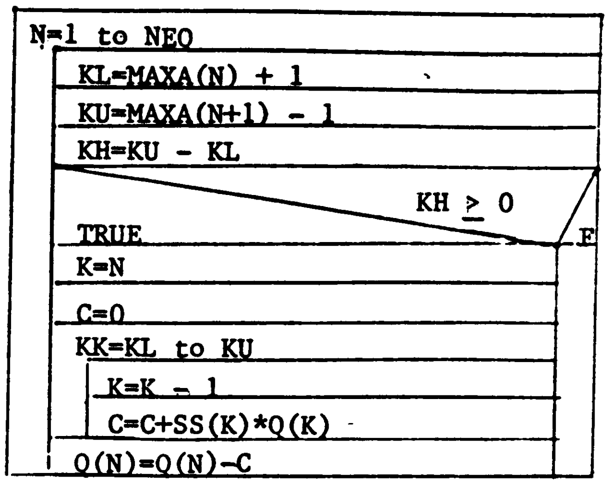
SKYLINE



SOLVE

```
N=1 to NEQ
    KN=MAXA(N)
    KL=KN+1
    KU=MAXA(N+1)-1
    KH=KU-KL
    KH > 0
    TRUE                                        F
        K=N-KH
        IC=0
        KLT=KU
        I=1 to KH
            IC=IC+1
            KLT=KLT-1
            KI=MAXA(K)
            ND=MAXA(K+1)-KI+1
            ND > 0
            TRUE                                F
                KK=MIN0(IC,ND)
                C=0
                L=1 to K
                    C=C+SS(KLT)-C
                SS(KLT)=SS(KLT)-C
            K=K+1
    KH ≥ 0
    TRUE                                        F
        K=N
        B=0
        KK=KL to KU
            K=K-1
            KI=MAXA(K)
            C=SS(KK)/SS(KI)
            B=B+C*SS(KK)
            SS(k)=C
        SS(KN)=SS(KN)-B
    SS(KN)=0
    TRUE                                        F
        Print "Not Positive Definite"
        STOP
```

FACTOR



```
N=1 to NEQ
    KL=MAXA(N) + 1
    KU=MAXA(N+1) - 1
    KH=KU - KL
    KH ≥ 0
    TRUE                                    F
        K=N
        C=0
        KK=KL to KU
            K=K - 1
            C=C+SS(K)*Q(K)
        Q(N)=Q(N)-C
```

REDUCE

```
N=1 to NEQ
    K=MAXA(N)
    Q(N)=Q(N)/SS(K)
NEQ > 1
TRUE                                         F
    N=NEQ
    L=2 to NEQ
        KL=MAXA(N) + 1
        KU=MAXA(N+1) - 1
        KH=KU - KL
        KH ≥ 0
        TRUE                                 F
            K=N
            KK=KL to KU
                K=K - 1
                Q(K)=Q(K)-SS(K)*Q(N)
        N=N + 1
```

BACSUB