AN EVALUATION OF TURBO PASCAL AS A PROGRAMMING LANGUAGE FOR STRUCTURAL ENGINEERING

by

Kenneth Robert Nuttall

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

R.M. Barker

S.D. Johnson

December, 1986

Blacksburg, Virginia

AN EVALUATION OF TURBO PASCAL AS A PROGRAMMING LANGUAGE FOR STRUCTURAL ENGINEERING

by

Kenneth R. Nuttall

Committee Chairman: Siegfried M. Holzer
Civil Engineering

(ABSTRACT)

Turbo Pascal, a compiler for the personal computer, was investigated to determine if it is compatible with the calculations required by Structural engineers. The compiler was evaluated for calculation intensive programming as well as programming for data manipulation.

A processor and a preprocessor for a plane frame program were used because they test the power of the language for calculations and they require the software to be user friendly and highly flexible with regard to manipulation of data.

A description of the development and a user's guide to the programs are included.

Acknowledgements

To Dr. Holzer, for the hours he listened to me ramble on about my research.

To Dr. Barker, for giving my programs the acid test--releasing them to a class of students.

To my friends at the Wesley Foundation, for their love and support.

To my parents, Bob and Shirley, for their love and support.

And especially to my wife, Molly, for her love, care, support, and the year she spent as a 'puter widow so I could finish my research.

Table of Contents

Chapter 1 Introduction 1

Chapter 2 Turbo Pascal Basics 4
The Great Underlying Rule of Pascal 4
Turbo Pascal and Structured Programming 8
Limitations and Restrictions of Turbo Pascal Version 3.0 10
Graphics in Turbo Pascal 15
Data Structures in Turbo Pascal 22
The Main Menu 26

Chapter 3 Comparisons and Translations 29
Benchmark Tests 29
Accuracy 33
BASIC vs Turbo Pascal 35
FORTRAN vs Turbo Pascal 41

Chapter 4 Program PREP 46
Planning 46
Design of Program PREP 48

Chapter 5 Users' Guide for PREP and FRAME 61
Hardware Requirements 61
Setting Up to Run PREP or FRAME 61
Units 61
Running Program PREP 62
MAIN MENU 63
F1-Control Variables 64
F2-Member Incidence 65
F3-Joint Constraints 66
F4-Hinges 68
F5-Joint Coordinates 68
F6-Element Properties 69
F7-Load Data 70
Joint Loads 71
Member Actions 72
Prescribed Joint Displacements 73
F8-Display Frame 73
F9-Exit to FRAME 74
F10-Exit to DOS 75
Safeguards 75
Running Program FRAME 75

Chapter 6 Conclusions 79

References 83

Appendix 1 Problem Assignments and Solutions from Microcomputer Applications in Civil Engineering 84
Problem Set 2 85
Problem Set 3 92
Problem Set 4 107
Problem Set 5 118
Problem Set 6 125

Appendix 2 Computer Listings for Program FRAME 134

Appendix 3 Computer Listings for Program PREP 160

Vita 252

List of Tables

Table 3-1	Execution Time for PC Magazine's Benchmark Tests	31
Table 3-2	Personal Engineering & Instrumentation News's Benchmark Tests Results	32
Table 3-3	BYTE's Benchmark Test Results	32
Table 3-4	Results of the Guard Program	34

List of Figures

Figure 2-1	A Sample Turbo Pascal Program	6
Figure 2-2	Memory Use	13
Figure 2-3	Comparison of DISPOSE to MARK/RELEASE	16
Figure 4-1	Treechart for Program PREP	50
Figure 4-2	An N-S Diagram to Define the Function Keys	55

Chapter 1 Introduction

Programming to support engineering calculations can be time consuming and expensive. The language used must be well supported by the compiler or the interpreter or much time will be wasted.

Many languages, and many systems for each language, have appeared with the growing popularity of the personal computer. Some are interpreted and some are compiled. The interpreted language has advantages during the program development. Each program statement is translated into machine code as it is executed. Errors cause the program to stop execution at the statement causing the error. The offending statement is fixed by the user and the program is started over. Once the program is debugged, an interpreted language is very slow to execute because each statement still must be translated each time it is executed. A compiled language is usually more difficult to debug due to the lengthy compilation and linking procedures. Once compiled, the execution times are much faster than those of an interpreted language.

A language used in the development of engineering programs needs to be as easy to use and debug as the BASIC interpreter that is supplied with most personal computers at the time of purchase. Once the program is in general use, the language must have the speed and formula solution power

of a compiled language such as FORTRAN.

The objective of this thesis is to evaluate the compiled language Turbo Pascal, Version 3.0. Turbo Pascal, Version 3.0, is a compiled language that is coupled with a full screen editor. The development of programs within the Turbo Pascal environment is exceptionally fast as is the execution of the programs once they are developed.

To evaluate the language, program FRAME (8, pp. 337-360) was translated from the listed FORTRAN into Turbo Pascal and a preprocessor, called program PREP, was developed to prepare the data file required by program FRAME. Extensions to program FRAME include: hinges near member ends, geometric imperfections (element and joint), and plotting of the frame using interpolation functions (8, pp. 53-54) for the deformed structure.

A review of the literature yielded very little information on Turbo Pascal with respect to engineering applications. The information available is almost without exception for business or personal applications. Although the language is slowly gaining popularity within academic circles, few textbooks have been published about Turbo Pascal.

Chapter 2 is a description of Turbo Pascal. Published benchmark tests are presented and brief comparisons are made between Turbo Pascal and BASIC and Turbo Pascal and FORTRAN in Chapter 3. The development of program PREP is presented

in Chapter 4 and Chapter 5 is a user's guide for both program PREP and program FRAME.

Chapter 2 Turbo Pascal Basics

Learning Turbo Pascal is a lesson in organization. Pascal was written as a teaching language by Niklaus Wirth. It was designed to force the student into learning good programming skills. "TURBO Pascal closely follows the definition of Standard Pascal as defined by K. Jensen and N. Wirth in the Pascal User Manual and Report" (1, p. 1). Extensions such as graphics, random access files, and the overlay system have been added to the capability of the compiler. More procedures and functions have been added to the standard library as well (1, p. 2).

This chapter will include a general program description and a discussion of how Turbo Pascal relates to structured programming. The limitations and restrictions placed on Turbo Pascal, Version 3.0, as well as the graphics and data structures that are supported by Turbo Pascal will also be discussed. A description of the Main Menu will conclude this chapter.

The Great Underlying Rule of Pascal

Pascal requires that "all identifiers must be declared before they are used" (2, p. 8-3). This includes variables, constants, data types, procedures, and functions.

A Turbo Pascal program consists of the program heading, the declaration section, and the programming body. Figure

2-1 presents a sample Turbo Pascal program.

The program heading, which is optional, may include a list of parameters to be passed into the program. The declaration section is a list of the labels, constants, data types, variables, procedures, and functions. They may be declared in any order and may appear more than once.

Labels are either number or character identifiers which are used with the direct branching statement, the GOTO statement. The GOTO statement was not used by the author since it allows more than one point of exit from a programming block. A structured programming block has only one point of entry and one point of exit. Multiple points of exit from a programming block makes the program difficult to follow and edit.

Constants are variables that will not change value throughout the program. The constant's type is determined by the compiler when the value is assigned. The programmer does not need to declare the constant's type. When a constant is typed, its value may be changed later in the program. Typed constants can be used to initialize variables.

Turbo Pascal does not limit the programmer to integer, real, or character data types. The programmer can define his own data types. The programmer must define data types to pass arrays and sets into the procedures.

The variable declaration also defines the variable

```
PROGRAM Sample;                                    {Program heading}

{This program receives three numbers
 to store in an array.  The largest
 of the three numbers is then returned
 to the screen.}

CONST                                              {Declaration section}
   A=100;

TYPE
   Group = array [1..3] of real;

VAR
   input, biggest  : real;
   i               : integer;
   B               : Group;

FUNCTION Max (s,t : real):real;
   BEGIN
      IF s >= t
         THEN Max := s
         ELSE Max := t;
   END;

BEGIN                                              {Programming body}
   WRITE('Enter a number:');
   READLN(input);
   biggest := input;
   B[1] := input;
   FOR i:= 2 to 3  DO BEGIN
      WRITE('Enter a number:');
      READLN(input);
      biggest := Max (biggest,input);
      B[i] := input;
   END;
   WRITELN('Your array is:  ', B[1], B[2], B[3]);
   WRITELN('The largest number in your array is ', biggest);
END.
```

Figure 2-1. A Sample Turbo Pascal Program.

type. The variables are listed and separated from their type by a colon. All of the variables must be defined or a compilation error occurs.

Procedures and functions must be defined, i.e., listed, or forward declared before the first statement that calls them. The procedure or function may be forward declared by using a copy of the procedure's heading followed by the key word FORWARD. The procedure or function will consist of the procedure heading, the declaration section, and the procedure body. The procedure heading contains the list of parameters, declared with their types, that are to be passed between the program and the procedure. Variables may be declared within the procedure or function and variable names which have been defined in the calling block may be redefined. They then become local variables in the procedure or function. Unless the tag VAR appears before the variables in the parameter list, the parameters passed into the procedure become local variables. Any operations that change the value of a local variable which appears in the parameter list does not affect the value of the variable in the calling program block.

The programming body is the group of statements that function as the working section of the program. The programming body starts with the key word BEGIN and concludes with the key word END. With the exception of the final END

statement, which is followed by a period, all Turbo Pascal statements are followed by a semicolon. Included in the programming body are control structures, assignment statements, and commands. If more than one command or assignment statement is needed within a control structure, the control structure will use a BEGIN and an END statement also.

Turbo Pascal and Structured Programming

Structured programming is a philosophy of programming based on the principle that the human mind is limited in the amount of information that it can efficiently process (8, p. 382). The program is broken up into smaller pieces called procedures and functions. The procedures and functions perform one or more related operations using control structures.

There are three types of control structures: sequence, alternative, and loop. The sequence structure is a block where the statements are performed in sequence. The alternative structure is a choice based on a condition. Turbo Pascal supports two alternative structures, the IF-THEN-ELSE and the CASE OF. The IF-THEN-ELSE structure provides a choice between two programming blocks while the CASE OF structure allows the choice to be made from two or more blocks of statements.

Turbo Pascal supports three loop structures. WHILE DO,

FOR DO, and REPEAT UNTIL. The WHILE DO loop performs the block of operations as long as the given condition is true. A FOR DO loop will perform the block of operations a set number of times according to the parameters assigned to the counter variable of the loop. The REPEAT UNTIL loop structure tests the given condition at the end of the block. This loop structure is used when the operations of the block must be performed at least once. Control structures direct the flow of the program so that a programming block only has one point of entry and exit (8, p. 329).

> Pascal is a structured language, which means that it is easier to write your program in modules by following certain, predefined steps. Certain parts of your program must be placed in certain locations within the program, and must follow certain conventions (2, p. 2-5).

The compiler of a structured language keeps track of the necessary control structures within the block. This eliminates the need for complicated GOTO branchings found in many nonstructured languages to emulate the required control structures. The danger in branching to another section of code with a GOTO statement is that the flow of the program may be impossible to trace through the code when attempting to debug the block of code.

Turbo Pascal also lends itself to top down programming. Top down programming is a method that starts with a main control module and branches down through levels of proce-

dures until all the tasks required by the program are performed. The main control module usually keeps track of the global variables and directs passing them into the first level of procedures and functions. These procedures either perform a specific task or call other procedures that perform specific tasks. The cycle repeats itself until all the tasks have been defined in a procedure or a function. The programs PREP and FRAME (Appendix 2 and 3) are examples of top down programming.

Limitations and Restrictions of Turbo Pascal Version 3.0

Turbo Pascal has been designed to work in the personal computer environment. Phillippe Kahn and his employees at Borland International have placed certain restrictions on Turbo Pascal that adapts the language to the environment in which it is used. Among these restrictions are: a 64K limit on the size of a source file, a 64K limit on the size of an object file, and a 64K limit on the size of the Data Segment that the compiler will access.

The design of the editor restricts the size of the source file. The 64K limitation includes all the comments that are distributed throughout the code. Turbo Pascal has allowed the programmer to get around this restriction by using included files. An included file is a separate file on a disk that contains procedures in source code. A com-

piler directive is placed in the code of the main program to call the included file. When the compiler directive is encountered at compilation, the disk, referenced by the optional drive designation, is searched. When the included file is found, it is compiled along with the main program file. If the included file is not found a compilation error occurs.

Since the code segment register of the 8086/8088, 80186, or 80286 processor can only address 64K, the object file is restricted to this size (6, p. 85). However, the restriction on the object file size may be circumvented by the use of overlay procedures. When a procedure is declared an overlay procedure, the compiler will store the compiled overlay procedure on the disk that contains the main program. Each of the consecutive overlay procedures are stored in the same file. The compiler then allocates a section of the memory to receive the pre-compiled code. As the overlay procedures are called, they are loaded from the disk into the reserved section of the memory and used. The compiler then frees the section of memory for use by the next called overlay procedure. The file which contains the overlay procedures may also be as large as 64K. The penalty for using overlay procedures is the increased time of execution due to the accessing of the disks. The author, and others who have used the program PREP, have not found this to be a

severe handicap. The access speed may be increased by using a hard disk or by using a RAM disk.

The only limitation to Turbo Pascal that can not be bypassed without the use of another type of data structure is the 64K size limit of the Data Segment. The Data Segment is the section of the memory that stores the values assigned to the variables declared in the program. The values assigned to the variables in procedures or functions are placed in the stack, as described below. When using Turbo-87, the Turbo Pascal compiler that supports the 8087 co-processor chip, a matrix that is 90 real elements by 90 real elements, a stiffness matrix for a plane frame consisting of 30 free joints, almost completely fills the Data Segment.

The stack is where the local variables from the procedures and the functions are stored. The heap is the dynamic memory for use by the program

The stack and the heap utilize the remaining memory after DOS, the program, and the Data Segment are stored. The segments in the Memory Use (see figure 2-2) that are stored under the stack/heap usually take up approximately 83K of the memory. "A program running under MS-DOS 2.0 on a 512K system has over 430K of memory in the heap,..." (2, p. 17-5). As figure 2-2 shows, the stack is assigned from the top of memory down towards the heap. The heap grows from the Data Segment, which contains the global variables, up-

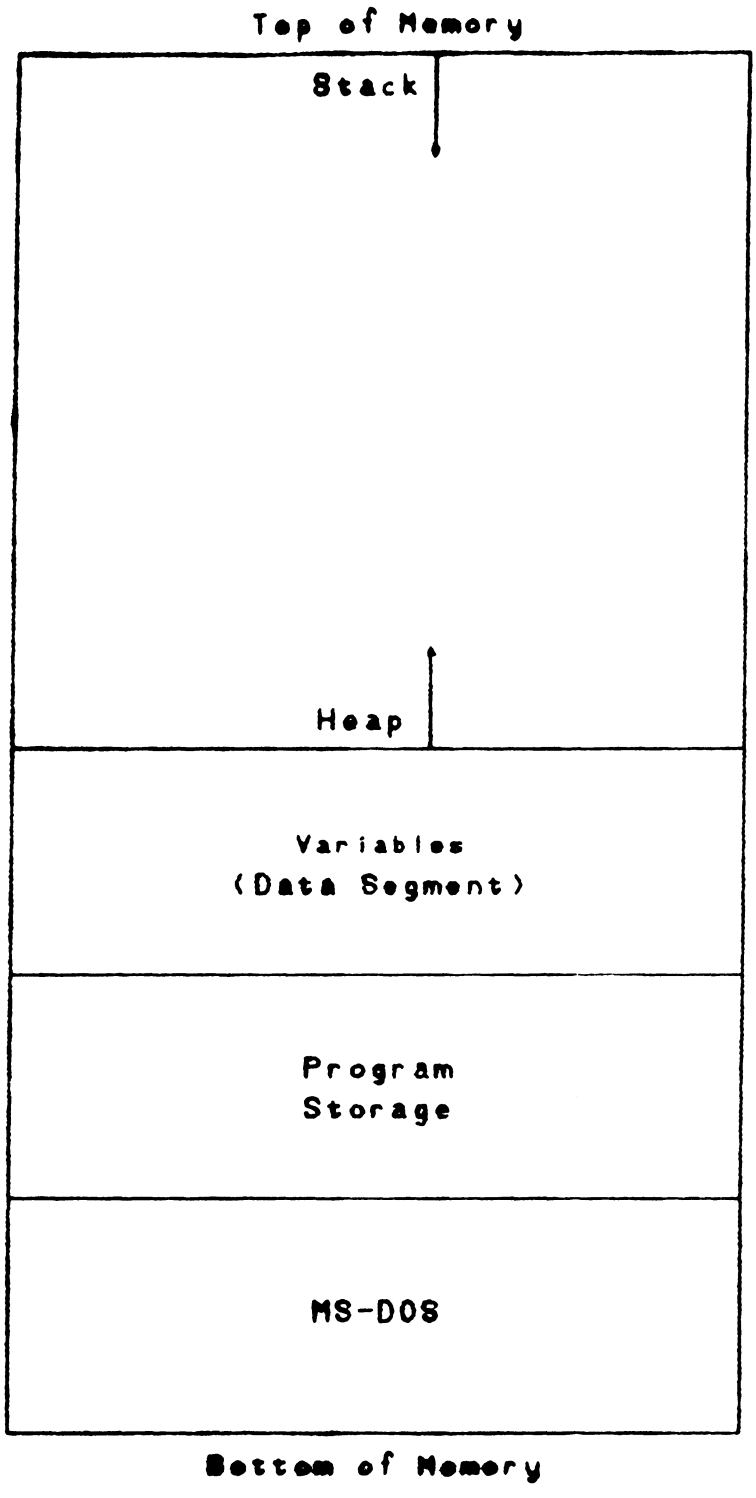


Figure 2-2. Memory Use (6, p. 87).

wards to the stack. The Turbo Pascal compiler keeps track of the sizes of the stack and heap and will issue an error statement if the stack and the heap collide. This collision can readily occur when many procedures are used in a program that uses several pointers or another data structure called the linked list.

There are memory management commands available to the programmer that allow him to create and destroy the data structures stored on the heap. The heap management is entirely the responsibility of the programmer. Maintenance of the heap pointer, which keeps track of the first empty address on the top of the heap, is the only operation that the compiler will perform to manage the memory. The Turbo Pascal function MEMAVAIL will return the amount of space remaining on the heap in 16 byte chunks called paragraphs.

The memory management commands NEW and GETMEM perform similar functions; they allot more memory on the heap to be used for the pointer variable that is declared as an argument of the command. The difference in the commands is the control of the memory allotted. The Turbo Pascal compiler determines the amount of space to reserve on the heap for the data structure by the type of pointer variable which is given as the argument of NEW. When GETMEM is used the programer must supply not only the pointer variable as an argument but also the number of bytes required to be placed

on the heap.

The other memory management commands, DISPOSE, FREEMEM, and MARK/RELEASE, are used to reclaim memory from the heap. DISPOSE and FREEMEM perform the opposite tasks from NEW and GETMEM. The Turbo Pascal compiler will free only as much memory as the pointer variable has been declared to hold when DISPOSE is used. The programmer must declare the pointer variable and the number of bytes associated with it when FREEMEM is used. The MARK/RELEASE command is used to free a large block of memory. It is best used when the programmer is creating a large data structure that will be used and then discarded. The command MARK is used to indicate the first record to be freed. All memory addresses from the marked record on will be freed for further use when RELEASE is called. The argument for both MARK and RELEASE is an arbitrary variable of the type pointer to an integer (^integer). Figure 2-3 shows a comparison of DISPOSE and MARK/RELEASE. DISPOSE and MARK/RELEASE should never be used together.

Graphics in Turbo Pascal

Standard Pascal omits graphics procedures since there were no personal computers in widespread use in 1970. Turbo Pascal has incorporated a great deal of graphics capabilities into the MS/PC DOS implementations. The system must be

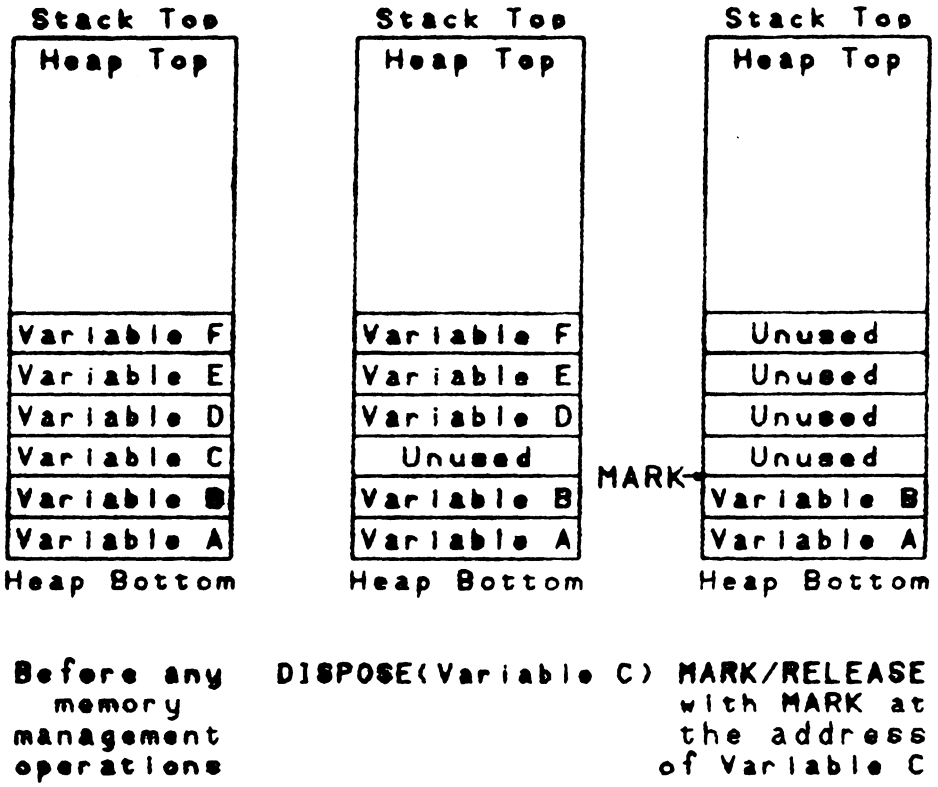


Figure 2-3. Comparison of DISPOSE to MARK/RELEASE.

completely IBM compatible to use the graphics procedures. The other systems, CP/M-86 and CP/M-80, that have versions of Turbo Pascal written for them, do not have the graphics capabilities of the DOS system.

The graphics capabilities include: screen mode control, windows for text and graphics, basic and extended graphics commands, and Turtlegraphics. Turtlegraphics, which was developed to allow the programmer to avoid cartesian coordinates, are not covered in the following discussions.

Screen mode control utilizes the following four modes: TEXTMODE, GRAPHCOLORMODE, GRAPHMODE, and HIRES.

TEXTMODE is available with 25 lines of 40 or 80 characters. The choice is also available between color and black and white screens when using a computer equipped with a color card and a color monitor. The key word TEXTMODE followed by an argument list, which consists of an integer variable from zero to three, will call the desired text mode.

The command TEXTCOLOR(I) is used to change the color of the text when a color mode has been chosen. The argument, I, can be an integer expression from one through 31. The integers zero through 15 correspond to the colors black, blue, green, cyan, red, magenta, brown, light gray, dark gray, light blue, light green, light cyan, light red, light

magenta, yellow and white. The integers 16 through 31 will cause the colors to flash. The arguement I can also be the name of the color and the word "blink" if the flashing letters are desired.

Turbo Pascal also allows the background of the text screen to be changed using the command TEXTBACKGROUND(I). For this command I is an integer ranging from 0 through 7 or the corresponding name of the color from the list above. There are several cursor commands to allow the programmer to design the desired screen. The commands WHEREX and WHEREY will return integers corresponding to the column and line numbers respectively. To place the cursor at a certain location, the command GOTOXY(I,J) is used. I is an integer expression which corresponds to the column number and J is an integer expression which corresponds to the line number of the desired cursor location.

GRAPHCOLORMODE is a medium resolution graphics mode which will address 320 vertical pixels by 200 horizontal pixels. Colors are chosen for use in this mode by the command PALETTE(I). The four available palettes each contain four colors. Three of the colors are set by the compiler and the fourth color, the background color, is chosen by the programmer using the command GRAPHBACKGROUND(I). The colors are chosen using the colors zero through 15 as listed above.

GRAPHMODE is the black and white equivalent of GRAPHCOLORMODE. On computers with color monitors, there is still a limited choice of color. The PALETTE command will allow a choice between two palettes.

The HIRES mode, which activates the screen to high resolution graphics, makes the screen 640 pixels wide by 200 pixels high. There is very limited color choice available to the programmer in this mode. The color of the background is black and the color to be plotted on the black background is chosen by the command HIRESCOLOR(I). I is the integer or the color name of the colors listed above. The color which is chosen then becomes color one for all the graphics commands listed below. When the color is changed by calling the HIRESCOLOR command, all the pixels that are currently plotted on the screen will be changed to the new color and the new color becomes color one for all further commands.

Windows on the screen are an excellent way to change just a small area of the screen while leaving the rest of the screen unchanged. The command WINDOW will define a window for use with text while the command GRAPHWINDOW will define a graphics window. To declare windows, the upper left hand corner and the lower right hand corner coordinates must be used as arguments to the window command. Once the window command has been given all further screen control commands become relative to the active window. The upper

left hand corner of the window becomes (1,1) if WINDOW has been called or (0,0) if GRAPHWINDOW has been called. Text windows leave the rest of the screen unaccessible. If the text entered becomes too long for the line defined in the window, the text will wrap around to the next line of the window. The text will also scroll up the window when too many lines have been entered. Graphics outside the window will be clipped which means that only the points plotted within the window will show up on the screen.

Basic graphics in Turbo Pascal contains just two commands: PLOT and DRAW. PLOT turns on the pixel at a given screen coordinate using a referenced color. This color depends on the palette which is chosen above when the screen is in medium resolution graphics. In high resolution graphics, the color argument must be set to one or the pixel is not plotted. DRAW will draw a line between two given screen coordinates. These arguments must be integers and must be within the range of pixels which correspond to the current screen mode for the point to be plotted.

Extended graphics use an included file, GRAPH.P. This file must be attached to every program which uses the extended graphics routines. There are ten extended graphics commands.

The command COLORTABLE supplements the Palette command. This command allows the four colors in the active palette to

be redefined by the order in which they appear in the argument list. The color table is used by the other graphics commands if the color argument is declared as -1, except for PUTPIC which always uses the color table.

The command ARC draws an arc starting at a screen coordinate, which is declared as an argument, through a given angle, with a given radius, in a specified color.

CIRCLE draws a circle at a given point on the screen of a specified radius. The location and radius are declared in the argument list of the command, and use a color specified by the programmer.

The commands ARC and CIRCLE create nearly perfect circles and arcs of circles when the screen is in a medium resolution graphics mode. The high resolution mode causes the arcs and circles drawn to be elliptical since the aspect ratio for many high resolution screens is equal to 0.4583333. The medium resolution screen has an aspect ratio of 0.9166667. A monitor with a screen aspect ratio of one would draw perfect circles and arcs.

GETPIC stores the image from a rectangular area of the screen. The buffer, which is one of the arguments of the command, must be sized by the programmer since the compiler will only perform the transfer of the picture into the buffer.

To copy the picture out of the buffer, the command

PUTPIC is provided. The picture area to receive the buffer's contents is defined using the lower left hand corner of the desired picture area.

The commands PATTERN and FILLPATTERN are used to define a pattern to be used and then places the pattern on the screen. This command is good for flashy computer displays.

Data Structures in Turbo Pascal

Turbo Pascal has several data structures that are useful to the engineer. The following data structures will be discussed: arrays, records, linked lists, and files.

"Simply put, an array is a collection of variables of identical type, each one of which may be referenced by a unique index value" (2, p. 13-1). An array in Turbo Pascal may be of any data type that has been defined. Brackets are used with arrays in Turbo Pascal. The array is defined by giving the first index number, two periods, and the final index number. This procedure is followed for each dimension of the array. Turbo Pascal stores arrays by rows with the rightmost index changing the fastest as the array is accessed. When an array is passed into a procedure, the address of the first element and the offset of the array (the length) is passed. The procedure then makes the required local copy of the array. These will become important factors in the discussion of the band solver which follows in

chapter three.

A record is a group of related data. The data is placed in the record in a component called the data field. The record is defined as a data type with each of the field names and their types in the definition. To reference the data field, the variable name that is of the record type is followed by a period and then the name of the data field. If several of the fields are to be filled at the same time, the programmer may elect to use the WITH statement. The syntax for the WITH statement is WITH (variable name) DO. After this statement is used, only the data field names are needed in the assignment statements. WITH statements may be nested to nine levels on a DOS system compiler.

Turbo Pascal has a record type called the variant record. This type follows all the rules of the regular record, but it has one data field called the tag field. This field may be filled with data of differing types which must be listed in the field definition of the record. Enough space is reserved in storage for each variant record, using the largest data type that has been declared for the variant field.

When records are passed into procedures, only the address and the offset are passed into the procedure. The procedure makes the local copy of the record, as it does with the array, to the stack.

Dynamic variables are variables that are allotted to the heap. They are called by using a pointer variable. A pointer variable is a variable that points to an address on the heap. The address on the heap contains the value of the variable in question. As stated above, program FRAME uses an array of pointer variables to store the global stiffness matrix for the plane frame analysis. The pointer just points to the place in the memory where the value is stored.

"A linked list or simply list is a sequence of nodes in which each node is linked or connected to the node following it" (11, p. 487). The linked list is a data structure which relies on the pointer variable. Linked lists are lists of records which have two or more data fields. One or more of these fields may contain pointer variables that point to other records in the list. If a list is singly linked, there is only one data field that contains a pointer to another record, usually the next record of the list. A doubly linked list will point to the record preceding and the record following for easy access up and down the list. Binary trees also use two pointer variable fields.

Turbo Pascal supports four types of files: sequential files, random access files, text files, and untyped files. All of the file procedures and functions available in Turbo Pascal will work on each of the four types of files.

ASSIGN associates an internal file name with a file

name that will be found on the disk. The ASSIGN statement must be used before any other file procedures or functions are used. CLOSE closes the file named in the argument list and updates the disk directory. To erase a file, the command ERASE is used. FLUSH forces a write of the file buffer to the disk or empties the buffer so a physical read from the disk takes place. READ/READLN reads one or more variables from the file. READ reads only the variables listed. READLN reads the variables listed and then moves the file pointer past the next end of line marker in the file. RENAME associates a new file name with the existing file name. The command RESET opens the file to be read. REWRITE opens the file to be written. In both of the above cases, the file pointer is placed at the first record in the file. This causes an existing record to be overwritten when REWRITE is used. SEEK positions the file pointer at the record of the file which is supplied as an argument to the command. WRITE/WRITELN writes the variables listed to the file. WRITELN also writes the end of line marker to the file. EOF, a boolean function, is true only if the file pointer is at the end of the specified file. FILEPOS returns an integer indicating the record number to which the file pointer is pointing. FILESIZE returns the number of records in the given file. FILEPOS and FILESIZE have equivalents with the word LONG attached at the beginning for

files that are longer than the maximum integer, which is 32,767. LONGFILEPOS and LONGFILESIZE return real numbers for the position and size, respectively.

The file operations are sequential by default. Each write operation writes to the end of the file. To read a record from the file, all the records which were written to the file before the desired record must be read.

Since Turbo Pascal files and records have been defined before they are used, any sequential file may become a random access file using the above command SEEK.

Many other languages including BASIC and FORTRAN create text files. Pascal, specifically Turbo Pascal, creates binary files on the disk. Turbo Pascal will support text files by the special definition TEXT. The text files that are stored on the disk can be read and used by other programming languages. Text files are good for large amounts of data which must be edited.

Turbo Pascal will also support untyped files. These files are just defined as FILE. They must be operated on in blocks of 128 bytes. The whole 128 bytes must be read or written at once. The author did not use untyped files.

The Main Menu

Turbo Pascal works from a main menu of 11 commands. The DOS commands to change the logged drive and the active

directory pathname are supported on the main menu. Entering L allows the logged drive to be changed while an A permits changes to the active directory. The work file, loaded into the compiler, is edited, compiled, run, or saved. To change the work file a W must be entered. "The M command may be used to define a main file when working with programs which use the compiler directive $I to include a file. The Main file should be the file which contains the include directives" (1, p. 16). To use the built-in editor the E command is entered. The command C compiles the source code of the work file while the command R runs the program. If the code has not been compiled, the command R compiles and runs the program. The compiler will load the main file before compiling or running the program. To save the results of the editing, the S command is used. The command D will display the directory of the logged disk drive. Turbo Pascal is exited by entering Q, the command to quit. There are three places that Turbo Pascal will place compiled code. The command O allows the programmer to enter the compiler options menu.

The default location of compiled code is the memory. The command M in the compiler options menu selects the memory as the location of the compiled code. To create a stand alone program the compiled program must be stored on the disk. A command file, which stands alone, is produced

by entering the command C from the compiler options menu. The third location that can be chosen from the compiler options menu is a chain file. The command H instructs the compiler to create a chain file. A chain file, like a command file, is compiled code stored on a disk. The chain file does not contain the Pascal library. Chain files are called from command files to extend the capabilities of Turbo Pascal beyond the 64K object code size or to link two programs together as the author did with programs PREP and FRAME. The compiler options menu has two additional helpful commands. The F command allows the programmer to find a run time error that occurred while using a command or chain file. The address of the error, which is given in the error message, is entered and the editor locates the offending statement in the source code. "The P-command lets you enter one or more parameters which are passed to your program when running it in Memory mode, just as if they had been entered on the DOS command line" (1, p. 192). The compiler options menu has a Q command that returns control to the main menu of Turbo Pascal.

Chapter 3 Comparisons and Translations

This chapter is included to give a comparison of Turbo Pascal with BASIC and FORTRAN, languages that are in general use by engineers today. Published benchmark test results will be cited and an accurracy test described. Later in this chapter, Turbo Pascal is compared to BASIC using problems assigned by Dr. Kamal Rojiani to his class, Microcomputer Applications for Civil Engineers. A comparison to FORTRAN will follow using a translation of program FRAME as developed by Holzer (8, pp. 337-360).

Benchmark Tests

The benchmark tests are designed to compare various languages, created for different uses, by a common measurement. The benchmark test are usually programs or procedures that measure the speed of the language being tested. Although no standard benchmark tests have been established, many leading computer science journals have designed and published their own tests.

PC Magazine uses the following tests: the Empty Loop, Integer Addition, Floating-Point Arithmetic, Character String Concatenation, Table Lookup, and File Access tests (5, p. 111). The Empty Loop test does nothing 10,000 times. The Integer Addition test counts from zero to 32,767. The Floating-Point Arithmetic test performs a loop 10,000 times

which multiplies and then divides two real numbers. The String Concatenation test joins two strings into one 10,000 times. The Table Lookup test fills an array with integers, then accesses the array 10,000 times. The File Access creates and opens a new file, writes the file of string records 132 bytes long, closes the file, re-opens the file, reads the file, modifies the records, writes them out again, and closes the file for the final time. The editors of PC Magazine purposely designed the File Access to be unfair to DOS or the languages by using the 132 byte records (5, p. 111). Table 3-1 shows the results of PC Magazine's tests for various languages and compilers.

Personal Engineering & Instrumentation News has published two benchmark tests that test Turbo Pascal, BASIC, compiled BASIC, MS Pascal, and Pascal MT+ (12, p. 31). The first uses the Sieve prime-number generator for testing integer operations, and the second performs matrix multiplication using a 10x10 matrix for testing floating point calculations (12, p. 31). See Table 3-2 for the published data.

The benchmarks used by BYTE include the Sieve prime-number generator, a puzzle solving program, and a quick sort on a worst case array of 100 real numbers (16, p. 272). The results are presented in Table 3-3. Wadlow's results (16, p. 272) are obtained using an earlier version of Turbo

Table 3-1.--Execution Time for PC Magazine's Benchmark Tests (5, p. 142 & p. 118). (in seconds)

	Empty Loop	Integer Count	Table Lookup	File Access	Float. Point	String Concat.
Turbo Pascal	0.23	1.00	1.10	6.0	39.0	19.0
Microsoft Pascal	0.18	0.71	0.90	9.0	18.0	14.0
Profes- sional Pascal	0.23	0.70	1.00	N/A	33.0	9.0
WATCOM BASIC	15.00	275.00	175.00	40.0	275.0	140.0
Quick BASIC	0.00	1.00	25.00	38.0	10.0	25.0
True BASIC	3.00	14.00	325.00	50.0	27.0	16.0
IBM BASIC	0.00	1.00	25.00	38.0	10.0	25.0
Better BASIC	2.00	25.00	27.00	45.0	140.0	70.0

Table 3-2--Personal Engineering & Instrumentation News's Benchmark Tests Results (12, p.30)
(in seconds)

	Turbo Pascal	BASIC	Compiled BASIC	MS Pascal	Pascal MT+
Sieve	1.5	168	1.5	1.11	1.95
Matrix	1.7	14	1.0	1.5	9.9

Table 3-3--BYTE's Benchmark Test Results (16, p.272)
(in seconds)

Time for:	Sieve	Puzzle	Qsort
Turbo Pascal			
Compile to memory	0.8	5.7	3.4
Compile to COMFILE	2.0	3.6	4.0
Execute in memory	15.0	69.0	1.3
Execute COM file	15.0	69.0	1.1
IBM Pascal			
Execute EXE file	76.0	416.0	1.2
IBM BASICA			
Execute	1980.0		

Pascal.

Benchmark tests on Turbo Pascal, Version 3.0, for the puzzle solving program show an increase in speed to 65 seconds and an increase in speed on the Sieve test to 13.0 seconds (3, pp. 282-283).

Accuracy

Currently, there are no published benchmark tests that measure the quality of a compiler or interpreter. William Kahan, a professor at the University of California at San Francisco, has developed a program called Paranoia to test the arithmetic used by a system. Testing the arithmetic is a measure of the quality of a system, whether it is a compiler or an interpreter. The results of his study are in print. Listings of the portion of Paranoia which deal with guard digits for round-off error have been published (10, pp. 230-235). Table 3-4 presents the results when each is run on a DOS based computer.

In Table 3-4, ulp refers to the unit-in-the-last-place, radix is the base in which the computer works (radix 2, binary, and radix 16, hexadecimal, are the most common bases in computer use), precision is the number of significant digits used in the base radix, and width is the number at which adding one makes no difference in the answer; the answer is too large to be affected by an addition of one.

Table 3-4--Results of the Guard Program

```
GUARD.PAS

     Turbo Pascal
          Radix=      2.0000000000E+00
          Precision= 4.0000000000E+00
          Width=      1.0995116278E+12
          UlpOne=     9.0949470177E-13
     Add/subtract lacks guard digit, cancellation obscured.

     Turbo-87
          Radix=      2.00000000000000E+000
          Precision= 5.30000000000000E+001
          Width=      9.00719925474099E+015
          UlpOne=     1.11022302462516E-016
     Add/subtract has a guard digit as it should.


GUARD.BAS

     GW BASIC
          Radix=       2
          Precision= 24
          Fpwidth=    1.677722E+07
          UlpOne=     5.960465E-08
     Add/subtract has a guard digit as it should.
```

BASIC vs Turbo Pascal

"BASIC is the most widely used programming language for microcomputers" (14, Chapter 3). Software packages that come with most microcomputers include some version of BASIC. Many engineers working with microcomputers use BASIC. A course given at Virginia Tech by Dr. Kamal Rojiani teaches Civil Engineering students the fundamentals of BASIC through practical problems which are commonly found in the practice of Civil Engineering. The problem statements and listings in both Turbo Pascal and BASIC for CE 4980, Microcomputer Applications in Civil Engineering (Winter Quarter 1986), are presented in Appendix 1 for a comparison of the languages. Five of the six problem sets assigned are discussed below.

Problem Set 2 covers the syntax for writing equations and using the built-in functions square and square root. The assignment included writing programs to compute the distance between two points, the combined resistance of four resistors arranged in parallel, and the area and the radius of the inscribed circle of a triangle from the lengths of the three sides. The differences in syntax occur in the built-in functions. BASIC uses SQR for square root where Turbo Pascal uses SQRT. Exponentiation shows a great difference between BASIC and Turbo Pascal. BASIC supports exponentiation by using the carrot (^) between the base and its exponent. Turbo Pascal does not support exponentiation.

The user is responsible for writing a subroutine to perform exponentiation. The function SQR will compute the square of the argument in Turbo Pascal.

Finding the roots of an equation using the bisection method was an introductory problem in numerical analysis assigned by Problem Set 3. A channel design using Manning's equation, which applied finding the roots of an equation using the bisection method, and a grade assignment problem rounded out the problem set. The roots of an equation problem was assigned a transcendental function. BASIC and Turbo Pascal both use the same syntax for the exponential, e, and the trigonometric functions, EXP(x) and SIN(x), respectively. Turbo Pascal closely adheres to the Pascal language as written by Niklaus Wirth and includes only the sine, cosine, and arctangent trigonometric functions. The programmer is responsible for writing routines that perform all the other trigonometric functions. The procedure for solution follows Rojiani (14, Chapter 12). This problem also incorporates the idea of the convergence factor.

The Manning's equation flow problem uses the binary search method from the problem above. The function that requires the roots to be found becomes more complex. In comparing the listings, note the simplicity of the BASIC statement where exponentiation is required. The corresponding Turbo Pascal statement must include EXP((2/3*LN(hydrau-

lic radius)) in place of (hydraulic radius)^(2/3). This limitation is not significant if the programmer is willing to make a library of mathematical functions which are often used but not supported by Turbo Pascal.

The grade assignment problem forces the programmer into using IF THEN statements to convert numerical data into characters. The DATA statement is used in the BASIC version for input. There is no equivalent in Turbo Pascal for the DATA statement found in BASIC. Turbo Pascal was designed to be interactive or use files. The DATA statement in BASIC is a remnant of the days when card batch files were the only way programs could be run.

Problem Set 4 continues in numerical analysis with a problem concerning numerical integration and moves into design using a column allowable axial load problem. Simpson's Rule (14, Chapter 12) was used to perform the numerical integration. The listings show that the difference between BASIC and Turbo Pascal is structure.

The program in Turbo Pascal could have been written without using functions and procedures, but the coding is cleaner when they are used. The BASIC code requires careful reading and possibly a flow chart to allow a clear understanding of the flow of the program.

BASIC supports a STEP function when it enters a loop. Turbo Pascal does not support this function.

> The for...do loop in Pascal sacrifices convenience for a gain in power. You cannot specify a step value by which to change [the counter], as you can in most other languages. To do that, you have to create your own loop using one of the other loop constructs...(2, p. 10-7).

In Turbo Pascal the counter does not have to be an integer, it can be any defined data type but real. This includes a data type that the programmer defines. Some types that the programmer defines will not lead to a clear use of the STEP function so it is not supported. The Turbo Pascal program for this problem uses the WHILE DO loop and increments a self kept counter within the loop to substitute for the STEP function.

The axial load problem was designed to show how useful small programs, which do not need to iterate to find the solution, can be in design where the engineer is required to solve one or two calculations many times in his work. The BASIC listing is written to emulate structure. This allows the program to be read from the top down. The simplicity of the calculations that it performs makes the control transfers easy to understand and follow.

Linear regression analysis using the least squares method was the only problem in Problem Set 5. Linear regression analysis is covered in Rojiani, (14, Chapter 10), which includes a listing of Rojiani's solution to Problem Set 5. Rojiani's solution in BASIC is more sophisticated

than the problem statement required. The Turbo Pascal version of the linear regression problem uses the Turbo Pascal subroutines which are called procedures. These procedures make it longer than the BASIC solution but allows the reader to inspect each task individually.

Problem Set 6 covered a variety of topics in the requirement of generating a simple xy graph. The programming required the following tasks: reading data from a data file, using arrays to store N points generated from a user defined function, and writing subroutines to perform the above and the necessary calculations for plotting the graph to the screen.

There are several differences in the listings when comparing the subroutines of BASIC to those of Turbo Pascal. The order of the subroutines is the first apparent difference. BASIC allows the subroutines to be placed anywhere in the program listing as long as they are not in a line of code which will be accessed before the subroutine is called. Standard practice is to place the subroutines in order after the main program listing. Turbo Pascal requires the subroutines, called procedures, to either be listed before the statement which calls the procedure or at least be predeclared before it is called. The author has chosen to list his procedures' code before the main program body or the procedure which calls the procedure in question. This will

save space in the file which stores the program listing.

BASIC writes a file as a text file. Turbo Pascal writes the file to the disk as a binary file unless a text file is specified in the variable declaration of the file's name. Using the default file format promotes an efficient use of disk space. Turbo Pascal has added the text option to allow user-entered data. The predefined data type TEXT is equivalent to FILE OF CHAR, a file composed entirely of character variables, which is not available under Turbo Pascal (2, p. 18-7).

Dynamic dimensioning of arrays is the only significant difference between BASIC and Turbo Pascal with respect to arrays. The educational basis of Turbo Pascal demands that all aspects of variables, including array variables, be declared before the variable is used in the code. BASIC will allow an array to be dimensioned using a variable input by the user. Another data structure must be used by Turbo Pascal to achieve dynamic dimensioning.

Turbo Pascal and BASIC use very similar commands and the same screen coordinates for graphing in both the medium resolution and the high resolution modes. Two differences occur between the BASIC and Turbo Pascal graphics commands. Turbo Pascal will not allow relative coordinates to be used when plotting points or drawing lines. BASIC will only plot white on black when high resolution graphics are being used.

Turbo Pascal allows the programmer to choose the color that will be plotted on the black background. The color is chosen from the table of colors that BASIC also uses for adding color when working in text mode. Accessing the color chip through a commercial program will change the plotting color to black on black.

FORTRAN vs Turbo Pascal

The comparative listings between FORTRAN and Turbo Pascal are found in Appendix 2. The program involved is a program that evaluates a plane frame through the matrix displacement method. The FORTRAN program is developed in Holzer (8, pp. 337-360).

The program FRAME was enhanced as well as translated from FORTRAN. Procedures to handle the hinges near member ends, the accompanying modifications to the member code and the joint code, the prescribed joint displacements, and the plotting of the frame in its undeformed and deformed configurations were added to the FRAME program. Modification of the band solver was necessary due to differences in the way FORTRAN and Turbo Pascal handle arrays when passing them between subroutines.

The translation process from FORTRAN to Turbo Pascal is fairly simple and straight forward. The FORTRAN version of program FRAME was developed using the principles of struc-

tured programming even though the language is "not considered to be well suited for coding structured programs" (8, p.334). The elementary operations, addition, subtraction, multiplication, and division, have the same syntax. The differences in the languages occur in the number of built-in functions that each supports and the fact that FORTRAN was not designed as a structured language.

FORTRAN has several built-in functions that are omitted from Turbo Pascal. FORTRAN supports the following trigonometric functions: sine, cosine, tangent, cotangent, arcsine, arccosine, and arctangent. Turbo Pascal supports only the sine, cosine, and arctangent functions. The following FORTRAN functions have no built-in equivalents in Turbo Pascal: the common logarithm, hyperbolic sine, hyperbolic cosine, and hyperbolic tangent, the statistical functions error function and gamma function, complex conjugate, transfer of sign, largest and smallest values, positive difference, and the type conversions integer to real, and the complex number conversions. These functions were left out of the formal definition of Pascal since Pascal was intended to be a teaching language. The functions are easily written by the programmer and placed in a library that he creates and then attaches to his programs as an included file when the functions are needed.

FORTRAN 77 has added some of the structures which

allow structured programs to be written. To emulate the other structures still requires use of the GOTO statement. To translate a block of FORTRAN code that is not structured, like the LINPACK routines used by the FORTRAN version of the program FRAME, it is sometimes necessary to draw a flow chart of the FORTRAN block in question. A structured flow chart, called a Nassi-Schneiderman diagram, is then prepared from the flow chart of the FORTRAN code. The coding for the Turbo Pascal version of the block follows the N-S diagram. The problems encountered in translating program FRAME from FORTRAN to Turbo Pascal from this step on were due to dynamic dimensioning techniques available only in FORTRAN.

The 64K limit on the Data Segment became a major hurdle when translating program FRAME from FORTRAN into Turbo Pascal, since the frame program was translated to run using the 8087 math coprocessor. The difference in the internal data format is four bytes of storage per real number. The version of Turbo Pascal which accesses the 8087 math coprocessor uses an internal storage of eight bytes per real number. The version of Turbo Pascal which does not access the 8087 math coprocessor only uses a four byte real number in internal storage. When a large array is defined the memory necessary for storage of the largest matrix possible is reserved. The array in question is the 90 by 90 stiffness matrix from program FRAME. The matrix above takes up

63.28K of the data segment when the math coprocessor is used. If the coprocessor was not used the matrix would only use 31.64K of the data segment.

The translated version of the program uses the heap to store the stiffness matrix. A full 90 by 90 matrix of pointers is defined and reserved. As the size of the global stiffness matrix is determined, only the amount of storage space required is requested on the heap. The price paid for this dynamic dimensioning of the global stiffness matrix is that for every coefficient of the stiffness matrix there are 12 bytes of storage used instead of the usual eight for a real number.

The band solver of the FORTRAN program was a commercial program LINPACK. LINPACK used dynamic dimensioning techniques available in FORTRAN 66 which are not available in Turbo Pascal. The technique which led to changing the band solver was the ability of FORTRAN to dynamically dimension an array as it is passed from one subroutine to another. The array is passed by passing only one coefficient. FORTRAN will than copy the array, by column, until the end of the array is reached. As stated in Chapter 2, Turbo Pascal transfers the address and the offset of the array to the subroutine and the subroutine makes the local copy of the array on the stack. Also, the array is stored by rows rather than by columns in Turbo Pascal. The band solver is

now a translation of the band solver presented by Cook (4, p. 45) Grandin (7, pp. 419-434) presents another complete frame program with a similar band solver. Cook's band solver was chosen because it had been used previously by the author for an earlier version of program FRAME written in FORTRAN.

The Turbo Pascal version of program FRAME uses the graphic capabilities to plot two views of the frame to the screen. The frame appears with and without deformations caused by the load case being run. There are no equivalent screen controls or graphics capabilities in FORTRAN.

Chapter 4 Program PREP

Special planning must be made when writing large programs in any language, Turbo Pascal included. Program PREP (see Appendix 3) is a 4500 line program designed to create the data file necessary for program FRAME. The file created should not cause a run-time error to occur when program FRAME is called. This chapter includes discussions on the planning and the design of program PREP.

Planning

There are several questions to be answered before writing a program. The programmer must decide the objective of the program or, if specifications are given, he must make them as complete as possible. He must also decide what data structures to use and how to implement them, how many tasks each of the procedures is to handle, and some alternatives must be considered for all of these questions.

The objective of program PREP is to provide the user with a preprocessor which allows him complete freedom in editing the data file required by program FRAME. The preprocessor is a tool to be used by the engineer. It is not to limit him or force him into a prescribed routine. It allows him to create not just manipulate. The limitations of program PREP are the limitations of program FRAME.

The data structure for program PREP was the major

question to be answered. Several alternatives were considered: RAM disk files merged into a single floppy disk file upon completion of the editing session, intermediate floppy disk files merged into a single floppy disk file upon completion of the editing session, and linked lists written to a floppy disk data file upon completion of the editing session.

The RAM disk was removed as one of the possibilities since it is DOS dependent (version 3.0 or later). The user would have to set up his own RAM disks. This option requires many file operations which makes the program much longer. The program was written to be used on most systems that are now in place.

The intermediate disk files would have placed a great burden on the user since large amounts of space would be needed on the disk to write the intermediate files and then the final data file. Each problem would need to start with a nearly empty data disk. This method also would have been slow, needing many files read, closed, reopened, written to, and finally merged into one file. All of these actions would be performed on the relatively slow floppy disk drives.

The linked list became the proper choice because unlike the first two choices the size of the data used would not be restricted to the disk size available. The advantages of

the RAM disk are still present since the linked list is maintained on the heap. With the linked list in internal memory, access times are much faster than having the data maintained in external files. The only drawback to using the linked list is the lack of information supplied with Turbo Pascal by Borland International on the Turbo Pascal implementation of the linked list. Koffman (11) or Radford and Haigh (13) give much better introductions to the Turbo Pascal implementation of the linked list.

The grouping of tasks into procedures needs to be flexible yet still stay within some guidelines. Most guidelines on structured programming recommend a length of no more than one page per subroutine. However, a length of no more than two pages of code was chosen as a good size for procedures. The whole procedure is readily followed yet there is enough room for many operations on two pages. Exceptions to the length limitation were allowed when the extra statements were an integral part of the subroutine or when the size of the stack grew unusually large due to the addition of another level of subroutines.

Design of Program PREP

After choosing the linked list as the data structure, the decision to use a linked list for each type of data needed by the program FRAME was made. The lists were de-

fined by the number and type of data fields needed. Some of the lists were able to be combined because they used the same number and types of data fields. The following types of data were combined: hinge and joint constraint data, the element properties data, and the joint load and prescribed joint displacement data.

A modified top down approach was used to develop program PREP. The method used wrote and tested each branch of the program from top to bottom instead of writing and testing the program from the first level to the bottom level of the program as a true top-down approach would take (8, p. 332). See Figure 4-1 for the tree chart of program PREP. Each group of procedures that created and maintained a specific type of data was written and tested before a new group of procedures for another data type were started.

The branches of the program were written in the order in which the program FRAME calls the data. Program PREP was started as a single source file. The 64K limitation of the Turbo Pascal editor was reached when all the procedures up to and including the element properties were in place. When the editor limitation was reached each group of procedures, corresponding to a type of linked list, was separated into a separate included file. The final included files are as follows: MEMINS.PAS, TYPE1.PAS, JTCOORDS.PAS, ELPROP.PAS, LOADS.PAS, TYPE3.PAS, MACT.PAS, DRAWS.PAS, and ETDOS.PAS.

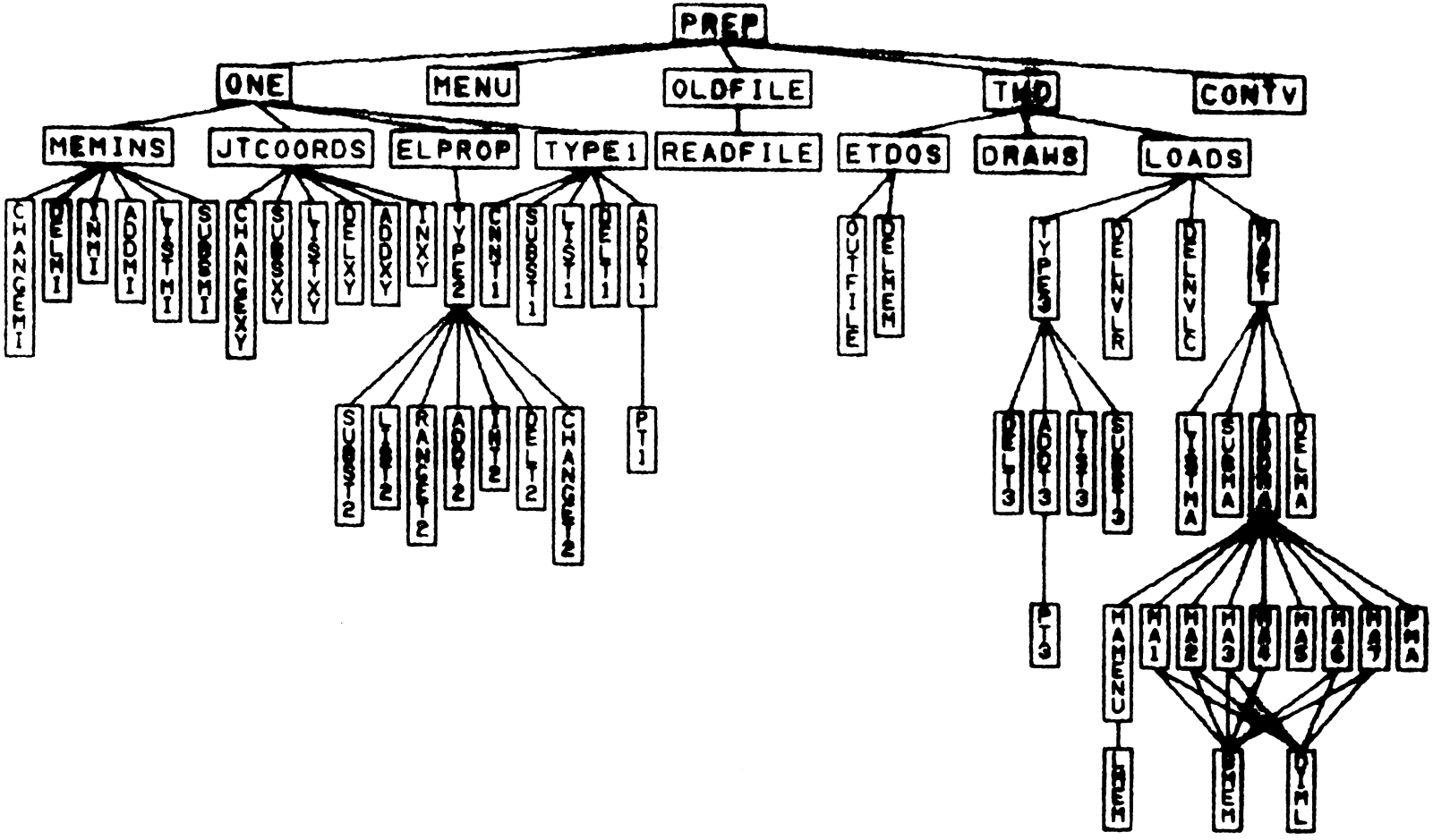


Figure 4-1. Treechart for Program PREP

Working with included files is very easy. When using the editor, the included file is declared as the working file and the file containing the main program is declared as the main file. When the program is to be compiled and run in memory, the compiler will load the main file if the run command is given when the included file is declared as the working file. No linking needs to be done by the programmer. The compiler, which is a one pass compiler, will do all the file manipulations. The code can be compiled to memory to be tested. After the final debugging the compilation toggle is set to create a command file or a chained file, if the program is to be called from another Turbo Pascal program. The included file is compiled in the object code with the main file.

Each of the included files in program PREP created and maintained a linked list as the data structure. The linked lists for member incidence, joint coordinates, and element properties are order dependent. Program FRAME will associate a number for a member or a joint to a record of the linked list without the record specifically stating the association in one of the data fields. The linked lists for hinges, joint coordinates, joint loads, member actions, and prescribed joint displacements are not order dependent. One of the data fields in the record being examined will state with which member or joint the rest of the data is associ-

ated.

Several of the linked lists are identical, i.e., the records have the same data fields declared in the same order. Hinge data and joint constraint data are combined in TYPE1. TYPE1 has a record that uses two integer data fields and a pointer field. The element properties, area, moment of inertia, modulus of elasticity, and coefficient of thermal expansion, all have records that contain one real data field and a pointer field. These linked lists are TYPE2 linked lists. Joint loads and prescribed joint displacements are combined as TYPE3. They include two integer data fields, a real data field, and a pointer field.

The member incidence records use two integer fields and a pointer field. Joint coordinates records consist of two real data fields and a pointer field. The member action records are the longest. They include two integer data fields, three real data fields, and a pointer field.

Subroutines have been written to perform the following operations on the linked lists: changing data in one of the records, deleting a record, adding a record, inserting a record, finding the location where a record is to be placed in the list, listing all the records to the screen, and placing the subheadings on the screen. The element properties routines include a routine that will add several identical records consecutively. The order dependent linked

lists allow changing the data in the records where the non-order dependent linked lists will not allow the data to be changed. Inserting records is allowed only with the order dependent linked lists. The non-order dependent linked lists are placed in order by the contents of their records. This ordering on the screen allows the user to input data in a comfortable order, not a forced order. The data is ordered on the screen for easy editing. The ordered list is the list that is written to the data file on the floppy disk.

Since each data type requires a separate set of procedures to create and maintain the linked list, the 64K limit on object code was reached during the development of the loading procedures. Overlay procedures were easily created. The "dummy" procedures ONE and TWO are compiled in a file called PREP.000. These dummy procedures are called through a case statement. The indicator variable used to call the dummy procedure is passed into the dummy procedure. The editing procedures are then called through another case statement using this same indicator variable. The procedures were split between the definition of the frame and the definition of the loading on the frame. Disk access occurs when a new overlay file is called. Splitting the procedures as described above places the disk access at a time when a major shift in thinking by the user is

occurring. The procedures to handle the main menu, the control variables, the input of the file name and date, and the reading of the previous data file are still in the main file.

Many menus are used in program PREP. The menus are coupled with the use of the function keys. Defining function keys is a simple case structure embedded in a REPEAT UNTIL loop. Figure 4-2 presents an N-S diagram showing the defining of the function keys. The function key generates a two character code. The first character is character 27, the character generated by the escape key on the keyboard. The second character code generated will be from 59 to 68. These codes correspond to characters on the keyboard. When coupled with the escape character they indicate one of the function keys. These coupled codes correspond to PF1 through PF10 respectively. Character codes 59 through 64 correspond to miscellaneous symbols. Codes 65 through 68 correspond to the capital letters A through D, respectively. The program uses the built-in function keypressed to determine if more than one character code is generated from the read statement. The call to read the keyboard is issued after the REPEAT UNTIL block is started. A case statement using the second part of the function key character code directs the flow of the program to the desired procedure. The use of the menus and the function keys provides the user

fk - a character variable to obtain input from the keyboard.
EscHit - a boolean variable to indicate the Escape character has been recorded.

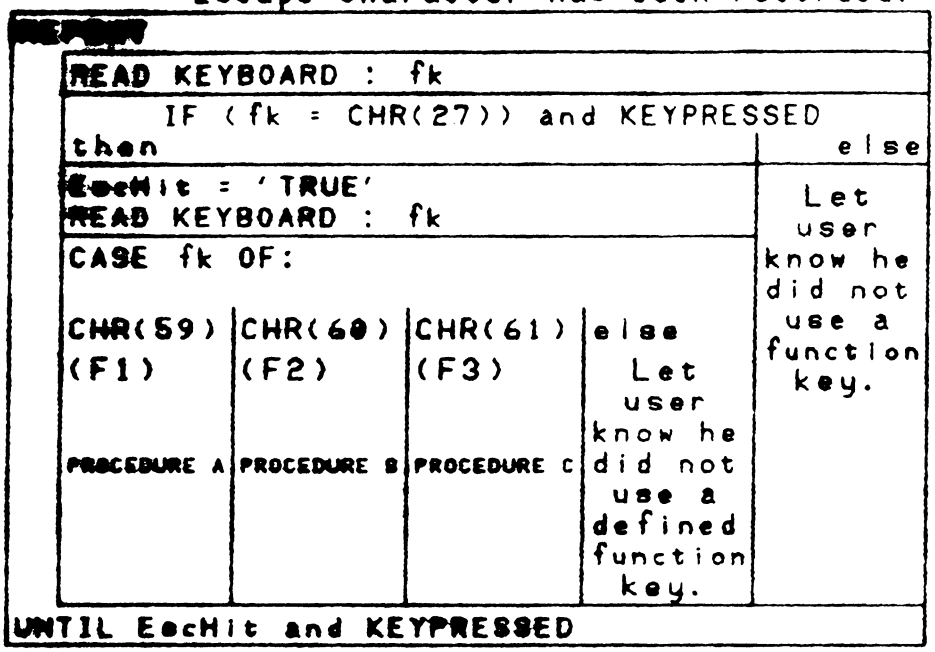


Figure 4-2. An N-S Diagram to Define the Function Keys.

with a one keystroke choice where all of the choices are grouped in one place instead of being spread out over the keyboard as with a letter menu.

The items in the main menu of PREP include: Control Variables, Member Incidences, Joint Constraints, Hinges, Joint Coordinates, Element Properties, Loads, drawing the frame to the screen, exiting program PREP to run program FRAME, and exiting program PREP to return to DOS. When procedures that maintain the linked lists are called from the main menu of PREP they too run on menus. The included files, depending on the type of linked list accessed, perform the operations described above: changing records, deleting records, adding records, and inserting records. The function key labeled F10 will always be a return to the menu which called the present menu except in the main menu where the user is returned to DOS.

The main menu and the use of the linked list as the data structure allows the user to reenter a data group one or more times. The list is easily maintained with counters for ordered data and accessing the pointer fields place non-ordered data into the list.

The simplest linked lists procedures were not used. Usually the required space for the new record is acquired from the top of the heap. The pointers are then changed to point to the new data record from the record which preceeds

it and from the new data record to the record which follows it. In PREP data is shuffled up and down the list as records are added and deleted. The last record of the data group will be the space that is freed when a record is deleted. The newly freed memory is closest to the heap and many times becomes the bottom of the heap. If the usual linked list procedures were used the dynamic memory could become fragmented. Fragmented memory will only allow the same type or a smaller type of record to be placed in newly freed memory. Placing the newly freed records as close to the free heap as possible helps to keep the memory from becoming fragmented. The author added to the complexity of program PREP by maintaining the linked lists used in this manner.

A pictorial menu is developed for the member actions. This allows the user to get a feel for the type of loading he is placing on the structure and the solutions to be expected from the program FRAME.

The same drawing routine is used in PREP that is used in program FRAME to plot the undeformed frame. The data structure usage had to be changed to match the linked lists of program PREP.

Bullet-proofing routines and loops were written to ensure a file that would not cause a run time error when used with program FRAME. The small loops around integer or real

number input directs the compiler to turn off the internal Input/Output checking routines. The program must then call the function IORESULT to determine if a type error was made when the user input the integer or the real number. If an error was made PREP beeps and loops back around the read statement to ask for the same input. This procedure will continue until the correct type of input is received. The small loop bullet-proofer keeps program PREP from prematurely halting due to an Input/Output error.

The bullet-proof routines and blocks of routines are written to protect program FRAME from a bad data file. These routines depend on the control variables input or read from the existing data file at the beginning of the program run. The data is forced to match the corresponding control variable. Order dependent linked lists have the control built in to the editing procedure. The user must add or delete data records until the control variable is matched. Exit from the routine is not permitted until the control variable is matched. The non-order dependent procedures have a section of code at the start of the editing procedure that searches the list for bad data. If bad data is found it is deleted.

The load cases needed special bullet-proofing routines since a piece of bad data could be embedded in the linked list. The other non-order dependent linked lists will only

have bad data at the end of the list. If too many load cases are present, the program will allow the user to return to the program's menu to change the number of load cases or look at the declared load cases. The alternative choice is to delete an entire load case. Placing a load on a joint or a member which does not exist, possible only when the control variable is reduced after the loads are declared, will cause the load record to be deleted. All of the load case counters are changed when records are deleted in this manner.

The program checks all of the linked lists before allowing the lists to be written to the disk. Any discrepancies are listed to the screen and the user is required to return to the main menu and fix the data before saving any of the changes.

When the lists are ready to be written to the disk two choices remain; the files may be written to the disk and control of the computer is returned to DOS or the file may be written to the disk and the chained program FRAME run. The chained program FRAME is found compiled in FRAME.CHN. This file can not run without being called from PREP which supplies FRAME.CHN with the Pascal library. The file name is passed from PREP into FRAME through the Data Segment. Both programs declare the file name as the first variable. When FRAME is chained, the Data Segment is not reinitialized

or cleared so the file name can be shared. Chained programs can pass any number of variables in this way. They must be the first variables declared and they must be declared in the same order.

Chapter 5 Users' Guide for PREP and FRAME

Hardware Requirements

The programs PREP and FRAME require the following hardware: an IBM PC, or compatible, with 256K (512K or more is recommended), the 8087 coprocessor chip and a graphics card.

Setting Up to Run PREP or FRAME

Boot DOS version 2.1 or later. PREP and FRAME have been compiled to run only under a DOS system.

Enter the DOS command GRAPHICS if a print out of the graphics screens from PREP or FRAME is desired. GRAPHICS is included on the DOS system disk.

Place PREP/FRAME Workdisk in the default disk drive. PREP uses an overlay file, PREP.000, which must be found on the disk in the default disk drive or a run-time error will occur. On most computers the default disk drive is the A> drive. On computers with hard disk drives PREP and FRAME should be copied to the hard disk or the default drive should be changed to the A> drive before loading the appropriate program.

Units

It is up to the user to keep the units consistent throughout both PREP and FRAME. The units to be used are: kips, inches, radians, and degrees Fahrenheit.

Running Program PREP

Function--to create or edit a data file for use with program FRAME.

To run program PREP enter PREP from the keyboard. As with program FRAME, an introductory paragraph will be written to the screen. Hit any key to proceed with the program.

After the screen is cleared, the prompt Enter File Name (1 to 8 characters): will appear. The drive designation may be entered along with the filename if the data file resides or is to reside on a specific disk. The file extension must not be entered or an I/O Error will occur at the end of the session when the file is to be written to the disk. The file extension .DAT will be added by the program to all file names. If the data file exists with an extension different from .DAT, rename the file or copy it to a file which has the extension .DAT.

The user will be prompted for the date. This date will be written to the file even if the file is only being inspected.

The program will search for the given file name with the .DAT extension. If the file is found, the statements Editing existing file. FILENAME.DAT was edited last on DATE. appears. DATE is the first piece of information found in the data file. The rest of the data file is read into

memory and the program advances to the MAIN MENU. If the given data file is not found, the message Creating a new file appears and the program starts with the Control Variable Menu. The Control Variable Menu will be described below.

The remaining sections of PREP are designed to be self explanatory. In general the responses to prompts will be: using the function keys, entering a character, entering an integer number or entering a real number. If a function key is to be used no return is required to answer the prompt. To answer the prompt for any other type of response the appropriate entry must be made and followed by a return. Loops are provided to protect against I/O Errors due to type mismatch. If the data is required to be in a range, the loop also checks the value entered against the corresponding control variable.

MAIN MENU

The main menu lists the groups of data needed to create a data file for FRAME. The function keys must be used to select the data group. The data groups are as follows: Control Variables, Member Incidence, Joint Constraints, Hinges, Joint Coordinates, Element Properties, Load Data, Display Frame, Exit to FRAME, and Exit to DOS.

F1-Control Variables

Control Variables initializes the main control variables, NE, the number of elements, NJ, the number of joints, and NLC, the number of load conditions or load cases. The specific control variable is chosen using a function key. The input for the variable is an integer followed by a return. If no data file exists, the procedure Control Variables is entered before the MAIN MENU is written to the screen. The Control Variables, NE, NJ, and NLC, must be the first to be defined so that the data may be checked by the subroutines as it is entered. This helps to create a file which will not cause a run-time error.

In the following data groups there are similar routines to handle the following chores: Change, Add, Insert, Delete, and Return.

Change allows the information within the record to be altered without breaking the order of the list of records. Change is available only for the order dependent data groups: Member Incidences, Joint Coordinates, and Element Properties.

Add attaches a new record to the end of the list of records.

Insert places a record within the list of records. The new record is inserted before the record the user inputs in

response to the prompt. Like Change, Insert is available only for the order dependent data groups.

Delete removes a record from the list of records. When working with the order dependent data groups only the record number is required to delete a record. PREP safeguards this delete by echoing the data in the record to be deleted and asking if the user still wants the record to be deleted. The random data groups--Joint Constraints, Hinges, and Load Data--require all the data field information of the record to be read so an exact match and only an exact match will cause a record to be deleted.

Return moves the user back one menu. If Return is invoked from one of the primary menus, F2 through F7, the number of records is compared with the corresponding control variable. Program control may not be returned from the primary menu to the MAIN MENU until the number of records matches the corresponding control variable.

F2-Member Incidence

The menu for Member Incidences includes: Change, Add, Insert, Delete, and Return.

Entry into Member Incidence causes a check for four conditions: no Member Incidence records, fewer Member Incidence records than the declared NE, more Member Incidence records than the declared NE, or the same number

of Member Incidence records as the declared NE.

When there are no Member Incidence records, i.e., a file is being created, the Add routine is entered until NE Member Incidence records are declared.

If the control variable NE has been changed such that more Member Incidences must be declared, the user is given the option of using the Add routine or the Insert routine.

Having more Member Incidence records than the control variable NE, forces the user into a loop which will delete records until only NE Member Incidence records remain.

After all of the above checks have taken place the menu for Member Incidence is written to the bottom of the screen. Function keys must be used to invoke the desired routine. The updated list of Member Incidence records will be written to the screen after each routine is called.

F3-Joint Constraints

Joint Constraints will force the user to declare at least one joint constraint by entering the Add routine if no Joint Constraint records exist.

The menu for Joint Constraints consists of Add, Delete, and Return.

Choosing Add from the Joint Constraint menu will cause another menu to be written to the screen. The choices are as follows:

F1- Fixed

All deflections at the joint entered will be constrained.

F2- Pinned

The deflections in the global 1 and 2 directions will be constrained.

F3- 1-3 Constraint

The deflections in the global 1 and 3 directions will be constrained.

F4- 2-3 Constraint

The deflections in the global 2 and 3 directions will be constrained.

F5- 1 Constraint

The deflection in the global 1 direction will be constrained.

F6- 2 Constraint

The deflection in the global 2 direction will be constrained.

F7- 3 Constraint

The deflection in the global 3 direction will be constrained.

These entries are made by the function keys and shown immediately on the list written to the screen.

To delete a joint constraint record all of the data fields must be matched exactly before the record will be

deleted.

F4-Hinges

When Hinges is entered the Add, Delete, Return menu will be written to the screen.

The Add sub-menu consists of:

F1-A end — a hinge is placed at the A end of the member.

F2-B end — a hinge is placed at the B end of the member.

F3-Both ends — a hinge is placed at each end of the member.

Like Joint Constraints, just using the function key causes the list of records to be updated.

To delete a hinge record all of the data fields must be matched exactly before the record will be deleted.

F5-Joint Coordinates

Joint Coordinates is an order dependent data group. It has checks similar to Member Incidences upon entry that depend on NJ instead of NE. The Change, Add, Insert, Delete, Return menu works the same as Member Incidences. The only difference between the two types of records is the format of the data fields within the record. Member Incidence works using integer data fields while Joint Coordi-

nates uses real numbers in the data fields. Joint Coordinates may be entered without a decimal point if the fractional part of an inch is not desired. The data will be right justified in the columns on the screen regardless of whether the decimal point is supplied by the user or the program.

F6-Element Properties

Element Properties is also an order dependent data group. The first sub-menu written to the screen contains:

F1 -AREA

F2 -MOMENT OF INERTIA

F3 -MODULUS OF ELASTICITY

F4 -COEFFICIENT OF THERMAL EXPANSION

F10-RETURN TO MAIN MENU

The Element Properties routine goes through the same checks as Member Incidence.

The initial Add or the Add caused by NE being greater than the number of Element Properties records will send the user to a loop which asks: Enter range: member i to member _. The variable i will be the next record to be declared and will be provided by the program. The response to the prompt is the largest member number that has the same property as i. It can be i or any member number greater than i and less than or equal to NE. The list of the

property will be updated and the prompt will reappear if the number of records does not equal NE.

The routines for Change, Add, Insert, and Delete are the same as the routines for Member Incidence except for the type of data handled. One real data field is used to handle each Element Property.

The Return in each separate Member Property menu will return the user to the Member Properties sub-menu. The return in the Member Properties sub-menu will allow the return to MAIN MENU only if all Member Properties record lists contain NE records.

F7-Load Data

If no Load Data records have been declared, PREP will enter the first load case and display the load case menu. Having only one load case will also place the user at this menu. The load case menu is as follows:

F1 -Joint Loads

F2 -Member Actions

F3 -Prescribed Joint Displacements

F10-End Load Case

Each function key calls the corresponding Load Data type. These data types are described below.

When the data file is read from a disk and there are less than NLC Load Data records, PREP will add load cases

until the number of Load Data records matches NLC.

After the number of Load Data records matches NLC the menu will read:

F1 -SELECT LOAD CASE

F10-RETURN

When F1 is chosen the next prompt will be Edit load case (1 to NLC):. The appropriate choice is made and the load case menu appears.

Joint Loads

The Joint Loads sub-menu contains Add, Delete, and Return. When Add is chosen this range prompt appears: Choose S for single load or M for multiples of the same load. An S will place the load at the joint number the user provides next. Entering M will cause the user to be asked for the first joint. The user will then be asked for the last joint to be loaded with the same load. After the direction of the load and the magnitude of the load are entered, the joint load records will be created so that the load is placed on each joint from the first joint declared through the last joint declared. The list of Joint Load records will be written to the screen.

Delete will delete the record input by the user in response to the prompts from the list of Joint Load records. The data fields must be declared as they are listed to be

deleted from the Joint Load list.

Member Actions

Member Actions uses a menu which includes Add, Delete, and Return.

Add will ask the same range question that Joint Loads asked. Then the screen will be cleared and a graphics menu will appear. This menu will show all the load types available for member actions. These load types include the following: Concentrated Lateral Load, Uniformly Distributed Lateral Load, Concentrated Axial Load, Uniformly Distributed Axial Load, Temperature Change, Element Imperfections, and Triangularly Distributed Lateral Load. The individual load type is chosen using the function keys. The screen will be cleared and the chosen load type data input graphics will be drawn to the screen. The load data is input in response to the prompts. The data will be placed on the drawing of the generalized member as each item is entered. The only exceptions to this are temperature changes and element imperfections where the data is only maintained on the command lines. The Add, Delete, Return menu will appear after each member action type is declared.

The Delete routine will only delete records which match the prompted delete data.

Prescribed Joint Displacements

When entering Prescribed Joint Displacements for the first time in each load case the user will be asked to enter PFAC. PFAC is the penalty factor to be used by FRAME when prescribed joint displacements are declared. (For more information on the penalty method see Bathe, K.J. 1982. Finite Element Procedures in Engineering Analysis. Prentice-Hall, Englewood Cliffs, New Jersey. p. 111-13.) The recommended PFAC is 1000. If the resulting displacement does not match the prescribed displacement a larger PFAC will be required to get an accurate solution.

Subsequent entries into Prescribed Joint Displacements will cause the user to be asked if he wants to change PFAC.

The Add, Delete, and Return routines work as described above in the section on Joint Loads.

F8-Display Frame

Display Frame will use the Joint Coordinates, the Member Incidence, and the Hinge data to plot the frame model on the screen. This is to allow the user a quick visual check of his model.

Warning: Hinges are shown as circles around the joint at the end where the hinge occurs. Since hinges are modeled as being near the members' ends, joints which have more than one member with a hinge at that joint should be checked

carefully to ensure that all hinges are modeled. No member information is given with the hinge display.

A copy of the screen is available using the <SHIFT> PrtSc keys if the DOS command GRAPHICS has been entered. To return to the MAIN MENU from Display Frame, hit any key.

F9-Exit to FRAME

Entering Exit to FRAME causes the data to be checked against the control variables. If any discrepancies occur the user is alerted and must change the data in the memory before PREP will call FRAME. PREP will indicate which data lists are keeping FRAME from being run.

Before FRAME can be called, PREP must write the data record lists to a disk and delete these lists from the memory of the computer. This is necessary so that FRAME has enough memory to pass the global stiffness matrix between subroutines. The user is told which operation PREP is working on by the messages: Writing file FILENAME.DAT. and Deleting records from memory. Control is then switched over to program FRAME.

The chained FRAME program follows the steps described below with the exceptions of writing the introductory paragraph to the screen and asking for the input file name. The input file name is passed from PREP into FRAME.

F10-Exit to DOS

Exit to DOS uses the same subroutines as Exit to FRAME. An IF-THEN statement which checks a parameter passed into the subroutine keeps Exit to DOS from calling program FRAME.

Safeguards

Many safeguards have been built into PREP to ensure a data file that will not cause a run-time error when FRAME is called. The loops which limit data to a range corresponding to a control variable and the check before Exit to FRAME or Exit to DOS are just two of them. The subroutines for Joint Constraints, Hinges, and Load Data have checks for joint numbers or member numbers larger than NJ or NE respectively. If Exit to FRAME or Exit to DOS directs the user to one of these data groups the offending data records will automatically be deleted when the appropriate group is called from the MAIN MENU. If any members or joints other than the last member or joint are to be deleted the corresponding joint constraints, hinges, and load data must be checked by the user to ensure the proper frame model is being evaluated.

Running Program FRAME

Function--to evaluate a frame model using the matrix displacement method. The input data, local element forces,

joint displacements, and joint forces are written to the output file FRAME.OUT. The frame model is plotted on separate screens with and without the deformations caused by the given loadings.

Program PREP should be run before running FRAME so that the input data file will be found. If the input data file is not found an I/O Error will occur and the program will be terminated. The purpose of program FRAME is to run, or rerun, data files which have been previously created. No provisions have been made to exit from program FRAME once it has been invoked. The standard <Control> Break interrupt will abort the program and search for COMMAND.COM on the default disk drive.

To start program FRAME just enter FRAME from the keyboard. The disk will be accessed, the screen will be cleared, and an introductory paragraph will be written to the screen. As instructed, strike any key to continue.

The program will ask for the input data file. The input file may reside on any disk drive. The required response to this inquiry is an optional drive designation followed by the file name, a period, and the extension. No directory path information may be input on this command line. For example, the workdisk is in drive A> and the input data file to be evaluated, FRAME1.DAT, is in drive B>. The correct response to Input data file: would be B:FRAME1.DAT.

Remember the file extension must be declared when running program FRAME.

The next direction required by the program concerns the disk drive onto which the output file, FRAME.OUT, is written. The question asked is: Write FRAME.OUT to the default drive (Y/N)? Here, as in all cases where character input is required, either upper or lower case letters may be used. If the output file is to reside on the workdisk then a 'Y' may be entered. If the file is to reside on a disk which is in another drive then an 'N' must be entered. The program will repeat this prompt until a 'Y' or an 'N' is entered. When an 'N' is entered the following question is asked of the user: Which drive do you want to write FRAME.OUT to: A, B, or C? Any legal drive designator may be used here. If the computer has two floppy disk drives then only 'A' or 'B' may be entered. If the computer has a hard disk installed then 'A' or 'B' or 'C' may be entered as the response. The default drive may be entered here if the user wishes.

FRAME.OUT will always be the filename and extension of the output file. The default drive, as discussed above, has the PREP/FRAME Workdisk inserted. If the file FRAME.OUT already resides on the disk which is designated to receive the output file, it will be written over. There is no file manipulation built into the program. All renaming or

copying to another disk or filename must be done from DOS before entering the program.

The program uses the matrix displacement method as presented by Holzer, S.M. 1985. Computer Analysis of Structures. Elsevier, New York. The band solver is presented by Cook, R.D. 1974. Concepts and Applications of Finite Element Analysis. John Wiley & Sons, New York.

The screen will be erased, the joints plotted and the members connecting the joints will be drawn. The frame will not show deformations and displacements at this time. Hit any key to proceed to the next screen.

The displaced and deformed frame will be drawn next. The deformations are calculated using eqns. 1.182-84 from Holzer's text, pp. 53-54. To proceed to the next load case hit any key. If the final load case is on the screen the program will end at this point.

FRAME.OUT will be closed and the program will return to DOS if COMMAND.COM is found on drive A> or drive C>. Otherwise the computer will require that a disk which has COMMAND.COM be placed in drive A> and any key hit before the computer can be used for any other program.

Chapter 6 Conclusions

Turbo Pascal has become a very popular programming language under many different applications. Its success is due to many factors. Turbo Pascal has held up very well in the benchmark tests. It rarely is the fastest compiled language yet it is consistently among the top languages tested.

Speed is not the only quality upon which to base a decision to use a language. The ease with which a program can be developed is another, sometimes more important, factor to be considered. The built-in editor that comes with Turbo Pascal is an excellent full screen editor. It emulates the popular word processing program WordStar. The author was familiar with WordStar before using Turbo Pascal so efficient use of the editor took little time to learn. During the development of the program, when the program is compiled in the memory and not on the disk, the built-in editor and the error locating functions within the compiler make Turbo Pascal as easy to use as an interpreted language. When an error is detected in the code during compilation the compiler returns an address of the offending statement. The error message is written to the screen and the editor starts to search for the location of the error. When it is found the editor waits for the programmer to strike the Escape key and then places the programmer in the code at the error.

The error messages are clear. (To save room in the memory, the error messages may be excluded from the run. The error number is given, as it is when the error messages are included, and may be looked up in the reference manual.) After the error is corrected the programmer goes through a series of four keystrokes to get the program to begin another run. Five keystrokes will save the corrected file and begin the next run. Compilation is quick. The 4500 line program PREP compiles in approximately two and one half minutes.

Turbo Pascal requires the programmer to use structured programming when writing code. This structure leads the programmer to writing code which is easy to follow and easy for other programmers to pick up and read. Documentation is still necessary but bad documentation can be overcome if the problem statement and the coding structures of the language are known. The control structures' syntax is very close to the wording that is used to describe the solution procedure. The choice as to which control structure to use thus comes naturally.

The Turbo Pascal compiler can not be surpassed in a cost/benefit ratio. The base version of Turbo Pascal, Version 3.0, is priced $69.95. The Turbo Pascal compiler that supports the 8087 math coprocessor, also Version 3.0, sells for $109.90 (1, p. 381). Turbo Pascal, Version 3.0, is

currently available in the University Bookstore for $35. Comparatively, the Microsoft Pascal Compiler, Version 3.3, costs $300 and the Professional Pascal, Version 2.3, costs $595 (5, p. 143). BetterBASIC costs $199, and the 8087 math support is an additional $49. The QuickBASIC Compiler costs $99. A newer version can be ordered through an educational discount for $49, but the compiler lacks any 8087 math coprocessor support. The BASIC Compiler from IBM costs $495 (5, p. 119).

The combination of Turbo Pascal's low price and interactive editor makes it the ideal language for the engineer. Although engineering firms may decide to pay more for a language compiler, they will probably hire someone to perform their programming. If engineers write their own programs, the Turbo Pascal compiler is less complicated to use and therefore the necessary software would be less expensive to develop.

Turbo Pascal is not without its drawbacks. As compared to FORTRAN, whether on a main frame computer or a personal computer, Turbo Pascal lacks many built-in mathematical functions. As stated earlier, the programmer can develop his own library of mathematical functions. Some mental gymnastics are required to circumvent the 64K limits placed on the source code, the object code, and the Data Segment. Version 4.0 of Turbo Pascal, which is to be released in

1987, is expected "to work around the 64K code barrier" (15).

Engineering students should learn several computer languages. Computer literacy is necessary in today's world of engineering design and analysis as more and more firms utilize personal computers. Turbo Pascal is an excellent language for the student engineer to learn and use. The programming skills and habits that are taught through use of the language will prove invaluable when applied to unstructured languages. Borland International has a very competitive product with students because of the low price which is charged and the power of the compiler.

FORTRAN may never be replaced on main frame computers but Turbo Pascal may replace it on the personal computer for engineering programming.

References

1. Borland International, Inc., TURBO Pascal Reference Manual Version 3.0, 2nd ed., Scotts Valley, CA, 1985.
2. Borland International, Inc., Turbo Tutor, 3rd ed., Scotts Valley CA, 1985.
3. Bridger, Mark, "Turbo Pascal 3.0: An Update on Borland's Compiler," BYTE, Vol. 11, No. 2, February, 1986, pp. 281-286.
4. Cook, R. D., Concepts and Applications of Finite Element Analysis, 1st ed., John Wiley & Sons, Inc., New York, 1974.
5. Dickinson, John, et al., "Benchmarking the Languages", PC Magazine, Vol. 4, No. 21, October 29, 1985, pp. 109-161.
6. Faulk, Ed, The Turbo Pascal Handbook, COMPUTE! Publications, Inc., Greensboro, North Carolina, 1986.
7. Grandin, Hartley, Jr., Fundamentals of The Finite Element Method, Macmillan Publishing Company, New York, 1986.
8. Holzer, Siegfried M., Computer Analysis of Structures, Elsevier, New York, 1985.
9. Jones, Jacqueline A. and Harrow, Keith, Problem Solving Using Turbo Pascal, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.
10. Karpinski, Richard, "Paranoia: A Floating-Point Benchmark," BYTE, Vol. 10, No. 2, February, 1985, pp. 223-235.
11. Koffman, Elliot B., Turbo Pascal a Problem Solving Approach, Addison-Wesley Publishing Company Inc., Reading, Massachusetts, 1986.
12. Quinn, Richard, "Turbo Pascal: A Powerful Tool for Technical Users," Personal Engineering & Instrumentation News, Vol. 3, No. 1, January, 1986, pp. 30-32.
13. Radford, Loren E., and Haigh, Roger W., Turbo Pascal for the IBM PC, PWS Publishers, Boston, 1986.
14. Rojiani, Kamal, Programming in BASIC for Engineers, PWS Inc., Boston, in print, 1986.
15. Sully, Archer, Technical Support, Borland International Inc., personal communication, Aug. 5, 1986.
16. Wadlow, Tom, "Turbo Pascal: A High Performance, Low-Cost Compiler," BYTE, Vol. 9, No. 7, July, 1984, pp. 267-278.

Appendix 1

Problem Assignments and Solutions from Microcomputer Applications in Civil Engineering

CE 4980 PROBLEM SET 2 Due: Jan. 27, 86

1. The distance between two points (x1,y1) and (x2,y2) is given by:

$$\text{Distance} = \sqrt{(x1-x2)^2 + (y1-y2)^2}$$

Write a program that uses INPUT statements to read in the coordinates of four points (x1,y1), (x2,y2), (x3,y3), and (x4,y4). Calculate and print the distance between points 1 and 2, 1 and 3, 1 and 4, 2 and 3, 2 and 4, and 3 and 4.

2. The combined resistance for N resistors arranged in parallel is equal to:

$$RT = \frac{1}{1/R1 + 1/R2 + \ldots + 1/RN}$$

Write a program to read four resistance values R1, R2, R3, and R4 and compute the combined resistance RT.

3. The area of a triangle is given by:

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$$

where a, b, and c are the lengths of the three sides and s is one-half of the perimeter of the triangle.

$$s = (a+b+c)/2$$

The radius of the inscribed circle r is given by:

$$r = \text{Area}/s$$

Write a program to read in the values of a, b, and c from a DATA statement and compute the area, and r. Print the input data and the value of area and r.

```
10 DEFDBL D,X,Y   'USING DOUBLE PRECISION ON ALL VARIABLES
20 CLS
30 PRINT "THIS PROGRAM FINDS ALL THE DISTANCES BETWEEN FOUR POINTS."
40 PRINT "YOU WILL BE ASKED TO INPUT THE X AND Y COORDINATES."
50 PRINT "THE DISTANCES FOUND ARE COMPUTED BY THE FORMULA:"
60 PRINT "DISTANCE = SQR((X1-X2)^2+(Y1-Y2)^2)."
70 INPUT "WHAT IS X1";X1:INPUT "WHAT IS Y1";Y1 'INPUT
80 INPUT "WHAT IS X2";X2:INPUT "WHAT IS Y2";Y2 'FROM
90 INPUT "WHAT IS X3";X3:INPUT "WHAT IS Y3";Y3 'THE
100 INPUT "WHAT IS X4";X4:INPUT "WHAT IS Y4";Y4 'KEYBOARD
110 PRINT "X1=";X1;"Y1=";Y1 'ECHO
120 PRINT "X2=";X2;"Y2=";Y2 'INPUT
130 PRINT "X3=";X3;"Y3=";Y3 'TO
140 PRINT "X4=";X4;"Y4=";Y4 'SCREEN
150 DIST12=SQR((X1-X2)^2+(Y1-Y2)^2) 'DISTANCE FROM P1 TO P2
160 DIST13=SQR((X1-X3)^2+(Y1-Y3)^2) 'DISTANCE FROM P1 TO P3
170 DIST14=SQR((X1-X4)^2+(Y1-Y4)^2) 'DISTANCE FROM P1 TO P4
180 DIST23=SQR((X2-X3)^2+(Y2-Y3)^2) 'DISTANCE FROM P2 TO P3
190 DIST24=SQR((X2-X4)^2+(Y2-Y4)^2) 'DISTANCE FROM P2 TO P4
200 DIST34=SQR((X3-X4)^2+(Y3-Y4)^2) 'DISTANCE FROM P3 TO P4
210 PRINT "THE DISTANCE FROM P1 (";X1;",";Y1;") TO P2 (";X2;",";Y2") IS";DIST12
220 PRINT "THE DISTANCE FROM P1 (";X1;",";Y1;") TO P3 (";X3;",";Y3") IS";DIST13
230 PRINT "THE DISTANCE FROM P1 (";X1;",";Y1;") TO P4 (";X4;",";Y4") IS";DIST14
240 PRINT "THE DISTANCE FROM P2 (";X2;",";Y2;") TO P3 (";X3;",";Y3") IS";DIST23
250 PRINT "THE DISTANCE FROM P2 (";X2;",";Y2;") TO P4 (";X4;",";Y4") IS";DIST24
260 PRINT "THE DISTANCE FROM P3 (";X3;",";Y3;") TO P4 (";X4;",";Y4") IS";DIST34
270 END
```

```
program  DistanceBetweenPoints  ;

var
     X1,X2,X3,X4              : Real;
     Y1,Y2,Y3,Y4              : Real;
     DIST12,DIST13,DIST14     : Real;
     DIST23,DIST24            : Real;
     DIST34                   : Real;

begin
    {**********  Input  **********}
    ClrScr;
    Writeln('This program finds all the distances between four points.');
    Writeln('You will be asked to input the X and Y coordinates of four points.');
    Writeln('The distance formula is DIST=Sqrt(Sqr(X1-X2)+Sqr(Y1-Y2)).');
    Writeln('  ');
    Writeln('  ');
    Write('Please enter X1  ');
```

```
    Readln(X1);
    Write('Please enter Y1  ');
    Readln(Y1);
    Write('Please enter X2  ');
    Readln(X2);
    Write('Please enter Y2  ');
    Readln(Y2);
    Write('Please enter X3  ');
    Readln(X3);
    Write('Please enter Y3  ');
    Readln(Y3);
    Write('Please enter X4  ');
    Readln(X4);
    Write('Please enter Y4  ');
    Readln(Y4);


    {********** Echo Input **********}
    Writeln('X1=',X1,'    Y1=',Y1);
    Writeln('X2=',X2,'    Y2=',Y2);
    Writeln('X3=',X3,'    Y3=',Y3);
    Writeln('X4=',X4,'    Y4=',Y4);


    {********** Compute the distances **********}
    Dist12:=Sqrt(Sqr(X1-X2)+Sqr(Y1-Y2));  {Distance from point 1 to point 2}
    Dist13:=Sqrt(Sqr(X1-X3)+Sqr(Y1-Y3));  {Distance from point 1 to point 3}
    Dist14:=Sqrt(Sqr(X1-X4)+Sqr(Y1-Y4));  {Distance from point 1 to point 4}
    Dist23:=Sqrt(Sqr(X2-X3)+Sqr(Y2-Y3));  {Distance from point 2 to point 3}
    Dist24:=Sqrt(Sqr(X2-X4)+Sqr(Y2-Y4));  {Distance from point 2 to point 4}
    Dist34:=Sqrt(Sqr(X3-X4)+Sqr(Y3-Y4));  {Distance from point 3 to point 4}

    {********** Print results to the screen **********}
    Writeln('The distance from P1 (',X1,',',Y1,') to P2 (',X2,',',Y2,') is ',Dist12);
    Writeln('The distance from P1 (',X1,',',Y1,') to P3 (',X3,',',Y3,') is ',Dist13);
    Writeln('The distance from P1 (',X1,',',Y1,') to P4 (',X4,',',Y4,') is ',Dist14);
    Writeln('The distance from P2 (',X2,',',Y2,') to P3 (',X3,',',Y3,') is ',Dist23);
    Writeln('The distance from P2 (',X2,',',Y2,') to P4 (',X4,',',Y4,') is ',Dist24);
    Writeln('The distance from P3 (',X3,',',Y3,') to P4 (',X4,',',Y4,') is ',Dist34);

end.  {**** end of the program ****}
```

```
10 DEFDBL R 'SETS ALL RESISTANCES TO DOUBLE PERCISION
20 CLS
30 READ R1,R2,R3,R4 'READS THE FOUR RESISTANCES FROM DATA STATEMENT
40 RT=1/(1/R1+1/R2+1/R3+1/R4) 'RT IS THE TOTAL RESISTANCE
50 PRINT "THIS PROGRAM COMPUTES THE TOTAL RESISTANCE OF FOUR RESISTORS IN PARALLEL"
60 PRINT "THE TOTAL RESISTANCE IS COMPUTED BY:"
70 PRINT "TOTAL RESISTANCE = 1/(1/R1+1/R2+1/R3+1/R4)"
80 PRINT "R1=";R1;"R2=";R2;"R3=";R3;"R4=";R4
90 PRINT "THE TOTAL RESISTANCE OF THESE FOUR RESISTORS IN PARALLEL IS";RT
100 DATA 100,800,4.5,1.2
110 END
```

```
program Resistances;
{*****************************************************************

     This program reads four resistances from the keyboard
     and computes the total resistance of a circuit with
     these four resistors in parallel.
     R1,R2,R3, and R4 are the resistances input.
     RT is the resistance of the circuit.
     RT=1/(1/R1+1/R2+1/R3+1/R4)

*****************************************************************}

var
     R1, R2, R3, R4, RT   :  Real;

begin
     {********** Initialize Values  **********}
     R1:=0.0;
     R2:=0.0;
     R3:=0.0;
     R4:=0.0;
     RT:=0.0;

     {**********  INPUT  **********}
     ClrScr;
     Write('What is R1?   ');
     Readln(R1);
     Write('What is R2?   ');
     Readln(R2);
     Write('What is R3?   ');
     Readln(R3);
     Write('What is R4?   ');
     Readln(R4);
```

```
    {********** ECHO INPUT **********}
    ClrScr;
    Writeln('The data you input is as follows:');
    Writeln(' R1= ', R1);
    Writeln(' R2= ', R2);
    Writeln(' R3= ', R3);
    Writeln(' R4= ', R4);

    {********** Calculation **********}
    RT:=1/(1/R1+1/R2+1/R3+1/R4);

    {********** OUTPUT **********}
    Writeln(' '); Writeln(' '); Writeln(' ');
    Writeln(' The total resistance of these four resistors ');
    Writeln(' in parallel =  ',RT);

end.
```

```
10 CLS
20 READ A,B,C 'READS THE LENGTHS OF THE THREE SIDES OF A TRIANGLE
30 PRINT "THIS PROGRAM COMPUTES THE AREA OF A TRIANGLE AND"
40 PRINT "THE RADIUS OF THE INSCRIBED CIRCLE."
50 S=(A+B+C)/2 'S IS HALF OF THE PERIMETER OF THE TRIANGLE.
60 AREA=SQR(S*(S-A)*(S-B)*(S-C))
70 RADIUS=AREA/S
80 PRINT "THE LENGTH OF SIDE A=";A
90 PRINT "THE LENGTH OF SIDE B=";B
100 PRINT "THE LENGTH OF SIDE C=";C
110 PRINT "THE AREA OF THE TRIANGLE=";AREA
120 PRINT "THE RADIUS OF THE INSCRIBED CIRCLE=";RADIUS
130 DATA 5.75,8.9,12
140 END
```

```
program Triangle;
{************************************************************************

     This program reads the lengths of the three sides of a
     triangle from the keyboard, computes the area of the
     triangle, and computes the radius of the inscribed circle.
     A, B, and C are the lengths of the triangle's sides.
     S is half of the perimiter of the triangle.  S=(A+B+C)/2.
     AREA=Sqrt(S*(S-A)*(S-B)*(S-C)).
     RADIUS=AREA/S.


************************************************************************}

var
     A,B,C,S,AREA,RADIUS  : Real;

begin
     {**********  Initialize Values  **********}
     A:=0.0;
     B:=0.0;
     C:=0.0;
     S:=0.0;
     AREA:=0.0;
     RADIUS:=0.0;
```

```
    {********** Input **********}
    ClrScr;
    Writeln('This program computes the area of a triangle with');
    Writeln('sides A, B, and C and then it computes the radius');
    Writeln('of the inscribed circle.');
    Write('What is the length of side A?  ');
    Readln(A);
    Write('What is the length of side B?  ');
    Readln(B);
    Write('What is the length of side C?  ');
    Readln(C);

    {********** Echo Input **********}
    ClrScr;
    Writeln('Your triangle is defined as follows:');
    Writeln('The length of side A is  ',A);
    Writeln('The length of side B is  ',B);
    Writeln('The length of side C is  ',C);

    {********** Calculations **********}
    S:=(A+B+C)/2.0;
    AREA:=Sqrt(S*(S-A)*(S-B)*(S-C));
    RADIUS:=AREA/S;

    {********** Output **********}
    Writeln(' '); Writeln(' '); Writeln(' ');
    Writeln('The area of your triangle is  ',AREA);
    Writeln('The radius of the inscribed circle is  ',RADIUS);

end.
```

CE 4980 PROBLEM SET 3 Due: Feb. 5, 86

1. Write a program to find the roots of an equation using the bisection method. Input to your program shall consist of the left and right-hand limits XA and XB, such that the function f(x) evaluated at XA and XB has opposite signs, and the tolerance . If the absolute value of the difference between two successive estimates of the root is less than then the current estimate of the root is acceptable.

   Use your program to evaluate the roots of:

$$f(x) = e^{-x} - \sin(-\pi x/2)$$

   The BASIC statement to evaluate the function is:

```
FX = EXP(-X) - SIN(-PI*X/2)
```

   You may use the following input values:

a) XA = 0 XB = 3, EPSILON = 1.0E-06
b) XA = 5 XB = 7
c) XA = 15 XB = 17

2. The discharge Q through an open channel of trapezoidal section is given by Manning's equation as

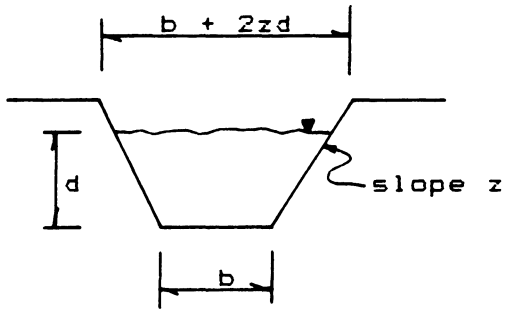
$$q = \frac{1.486}{n} A R^{2/3} S^{1/2}$$

in which Q = discharge, n = Manning's coefficient, A = channel area, S = bed slope and R = hydraulic radius. The hydraulic radius is given by

$$R = \frac{A}{(b + 2d\sqrt{z^2 + 1})}$$

where b is the bottom width, d is the depth and z is the side slope. The term $b + 2d\sqrt{z^2 + 1}$ is known as the wetted perimeter. The area of the trapezoidal channel is

$$A = (b + zd)d$$



Modify your program to determine the normal depth d corresponding to a required flow Q. Compute d for the following data

a) Q = 400, z = 1.5, n = 0.015, b = 10, S = 0.0002

b) Q = 250, z = 1.5, n = 0.015, b = 10, S = 0.0002

CE 4980 Problem Set 3

3. Professor Rojiani would like your assistance in assigning grades for his CE 4980 class. You are to write a program to read the name and score for each student in the class from DATA statements and determine a letter grade according to the following scale:

Score	Grade
100 - 90	A
89 - 80	B
79 - 70	C
69 - 60	D
< 60	F

The first DATA statement contains the number of students in the class. Your program should print a table consisting of the name, score and grade for each student. Use the following data:

```
400 DATA 11
410 DATA "PAUL BALSERAK", 70
420 DATA "WILLIAM CADDEN", 80
430 DATA "GREGORY CURREY", 90
440 DATA "CARL DUNCAN", 99
450 DATA "LISBETH FREEMAN", 95
460 DATA "TODD GREENWOOD", 85
470 DATA "DAVID HALL", 55
480 DATA "ROBERT HUNTGATE", 66
490 DATA "PAUL KNAEBEL", 78
500 DATA "KEVIN LONG", 88
510 DATA "HELEN STEYER", 91
```

```
10 CLS
20 '---------- DESCRIPTION OF PROGRAM ----------------------------------------
30 '        THIS PROGRAM USES THE BINARY SEARCH (HALF INTERVAL) METHOD
40 '        TO COMPUTE THE ROOTS OF THE EQUATION:
50 '        F(X)=EXP(-X)-SIN(-PI*X/2)
60 '---------- INITIALIZE CONSTANTS AND COUNTER ------------------------------
70 I%=1
80 PI=3.1415926549
90 EPSILON=.000001
100 '--------- FUNCTION DEFINITION ------------------------------------------
110 DEF FNF(X)=EXP(-X)-SIN(-PI*X/2)
120 '--------- INPUT --------------------------------------------------------
130 PRINT "THIS PROGRAM USES THE BINARY SEARCH (HALF INTERVAL) METHOD"
140 PRINT "TO COMPUTE THE ROOTS OF THE EQUATION:"
150 PRINT "F(X)=EXP(-X)-SIN(-PI*X/2)."
160 PRINT "PI IS DEFINED AS"; PI
170 PRINT "THE SMALLEST INTERVAL CONSIDERED, EPSILON, IS";EPSILON
180 PRINT "THE INPUT WILL BE FROM THE KEYBOARD."
190 PRINT: PRINT: PRINT :PRINT
200 PRINT "PLEASE ENTER THE ENDPOINTS OF AN INTERVAL YOU THINK CONTAINS A ROOT"
210 INPUT "WHAT IS XA";XA
220 INPUT "WHAT IS XB";XB
230 PRINT :PRINT :PRINT
240 PRINT "YOUR INTERVAL IS BETWEEN XA=";XA;"AND XB=";XB
250 '--------- FIND VALUES AT INTERVAL ENDS ----------------------------------
260 FXA=FNF(XA)
270 FXB=FNF(XB)
280 '--------- CHECK FOR OPPOSITE SIGNS -------------------------------------
290 IF SGN(FXA) <> SGN(FXB) THEN GOTO 330          'EMPTY TRUE BLOCK
300 PRINT "F(XA)="; FXA; "F(XB)="; FXB
310 PRINT "YOUR INTERVAL HAS THE SAME SIGN AT BOTH ENDS.  PLEASE TRY AGAIN."
320 GOTO 190                                      ' RETURNS FOR NEW INPUT
330 '--------- CHECK FOR ENDPOINT EQUAL TO ZERO ------------------------------
340 IF FXA=0! THEN GOTO 360
350 GOTO 380                                       ' EMPTY FALSE BLOCK
360 ROOT=XA  :  FROOT=FXA
370 GOTO 580                                       ' SEND TO OUTPUT
380 IF FXB=0! THEN GOTO 400
390 GOTO 420                                       ' EMPTY FALSE BLOCK
400 ROOT=XB  :  FROOT=FXB
410 GOTO 580                                       ' SEND TO OUTPUT
420 ' END IFS
430 '--------- ITERATIVE PROCEDURE ------------------------------------------
440 XM=(XA+XB)/2                                    ' MIDPOINT
450 FXM=FNF(XM)                                    ' FUNCTION VALUE AT MIDPOINT
460 '--------- CHECK FOR FXM EQUAL TO ZERO ----------------------------------
470 IF FXM=0! THEN GOTO 570          ' TRANSFER TO ROOT ASSIGNMENT STATEMENT
480 '--------- REPLACE ONE ENDPOINT WITH XM ---------------------------------
490 IF SGN(FXM) = SGN(FXA)  THEN GOTO 496           ' REPLACES XA
493 GOTO 500
```

```
496 XA=XM
498 FXA=FXM
500 IF SGN(FXM) = SGN(FXB)  THEN GOTO 506           ' REPLACES XB
503 GOTO 510
506 XB=XM
508 FXB=FXM
510 IF I%=1 THEN GOTO 530                          ' TEST ALL BUT FIRST ITERATION
520 IF ABS(LASTX-XM) < EPSILON THEN GOTO 560       ' TRANSFER TO ROOT ASSIGNMENT
530 LASTX=XM                                       ' SET LASTX FOR NEXT ITERATION
540 I%=I%+1                                       ' SET COUNTER FOR NEXT ITERATION
550 GOTO 430                                       ' RETURN FOR NEXT ITERATION
560 '---------  ROOT ASSIGNMENT  ------------------------------------------------
570 ROOT=XM  :  FROOT=FXM
580 '---------  OUTPUT  ---------------------------------------------------------
590 PRINT "IT TOOK";I%;"ITERATIONS TO GET AN ANSWER."
600 PRINT "ROOT=";ROOT
610 PRINT "F(ROOT)=";FROOT
620 '---------  USER CONTROL FOR ANOTHER ROOT  ----------------------------------
630 PRINT :PRINT
640 INPUT "ARE THERE OTHER ROOTS (Y/N)";ANS$
650 IF ANS$="N" OR ANS$="n" THEN GOTO 670
660 I%=1 :  GOTO 190            ' RESET COUNTER AND RETURN TO TOP OF PROGRAM
670 END
```

```
Program BinarySearchMethod;

const
    Epsilon=1E-06;


var
    XA, XB, XM, FXA, FXB, FXM, LASTX: real;
    ANSWER: char;

function F(X: real):real;
    begin
    F:=Exp(-X)-Sin(-Pi*X/2)
end;{of function F(X)}
```

```
function Sqn(FX:real):integer;
     begin
     if FX = 0 then begin
        Sqn:=0
     end
     else begin
        if Abs(FX)=FX
           then Sqn:=1
           else Sqn:=-1;
     end;
end;

begin
ClrScr;
writeln('This program finds the roots of an equation using the binary search');
writeln('(half interval) method.  The equation, found in the function F(X),');
writeln('is:  f(x)=exp(-x)-sin(-pi*x/2), where x will be entered from the,');
writeln('keyboard and pi=',Pi,'.');
writeln('The error tolerance epsilon=',Epsilon,'.');

repeat {until ANSWER = 'N' or 'n' (No roots to be found)}
    repeat {until Sqn(FXA) <> Sqn(FXB) (Estimates on either side of root.)}
         writeln; writeln;
         write('What is XA?  ');
         readln(XA);
         write('What is XB?  ');
         readln(XB);
         writeln; writeln;
         writeln('You have defined XA= ',XA,' and XB= ',XB);
         FXA:=F(XA);    {Call function F(X)}
         FXB:=F(XB);    {Call function F(X)}
         if Sqn(FXA) = Sqn(FXB) then begin
          writeln; writeln;
          writeln('FXA= ',FXA);
          writeln('FXB= ',FXB);
          writeln('ERROR: same sign. Please try again!');
        end;
    until Sqn(FXA) <> Sqn(FXB);
    if FXA=0.0 then begin
         XM:=XA;
         FXM:=FXA;
    end;
    if FXB=0.0 then begin
         XM:=XB;
         FXM:=FXB;
    end;
```

```
    if (FXA <> 0.0) and  (FXB <> 0.0) then begin
         XM:=(XA+XB)/2;
         FXM:=F(XM);   {call function F(X)}
         LASTX:=1800000000000000000000000000.0;
              {if using TURBO-87 w/ possibility of large root use 1E+308}
         if FXM <> 0.0  then begin
              if Sgn(FXA)=Sgn(FXM) then begin
                 XA:=XM;
                 FXA:=FXM;
              end
              else begin
                 XB:=XM;
                 FXB:=FXM;
              end;
              repeat {until Abs(LASTX-XM)<EPSILON or FXM=0}
                 LASTX:=XM;
                 XM:=(XA+XB)/2;
                 FXM:=F(XM);   {CALL FUNCTION F(X)}
                 if FXM <> 0.0 then begin
                    if Sgn(FXM)=Sgn(FXA) then begin
                       XA:=XM;
                       FXA:=FXM;
                    end
                    else begin
                       XB:=XM;
                       FXB:=FXM;
                    end;
                 end;
              until (Abs(LASTX-XM) < EPSILON) or (FXM=0.0);
         end;
    end;
    writeln; writeln;
    writeln('The root equals ',XM);
    writeln('The actual value at the root is ',FXM);
    writeln(' '); writeln(' ');
    write('Are there any other roots to be found (Y/N)? ');
    readln(ANSWER);
until (ANSWER ='N') or (ANSWER = 'n');
end.
```

```
10 CLS
20 COLOR 1,0,0
30 '
40 '--------  PROGRAM DESCRIPTION  ----------------------------------------------
50 PRINT "THIS PROGRAM USES THE BINARY SEARCH (HALF INTERVAL) METHOD"
60 PRINT "TO FIND THE DEPTH OF A TRAPEZOIDAL CHANNEL REQUIRED FOR A"
70 PRINT "GIVEN DISCHARGE USING MANNING'S EQUATION:"
80 PRINT "Q=1.486/N*A*R^(2/3)*S^(1/2)"
90 PRINT "WHERE Q=DISCHARGE REQUIRED, N=MANNING'S COEFFICIENT,"
100 PRINT "S=BED SLOPE, Z=SLOPE OF CHANNEL WALLS, B=MIN. WIDTH OF CHANNEL,"
110 PRINT "A=CROSS SECTIONAL AREA OF CHANNEL (=(B+Z*D)*D), AND"
120 PRINT "R=HYDRAULIC RADIUS (=A/(B+2*D*SQR(Z^2+1)))."
130 PRINT "YOU WILL DEFINE Q,Z,N,B, AND S FROM THE KEYBOARD."
140 PRINT "LATER YOU WILL BE ASKED FOR A RANGE YOU THINK D WILL FALL IN."
150 '
160 '--------  INPUT  ------------------------------------------------------------
170 PRINT :PRINT :PRINT :PRINT
180 COLOR 2,0,0
190 INPUT "WHAT IS THE MINIMUM CHANNEL WIDTH, B";B
200 INPUT "WHAT IS MANNING'S COEFFICIENT, N";N
210 INPUT "WHAT IS THE REQUIRED DISCHARGE, Q";Q
220 INPUT "WHAT IS THE BED SLOPE, S";S
230 INPUT "WHAT IS THE SLOPE OF THE CHANNEL WALLS, Z";Z
240 PRINT :PRINT :PRINT :PRINT
250 COLOR 3,0,0
260 PRINT "YOU HAVE DEFINED THE FOLLOWING:"
270 PRINT "MINIMUM CHANNEL WIDTH B=";B
280 PRINT "MANNING'S COEFFICIENT N=";N
290 PRINT "REQUIRED DISCHARGE Q=";Q
300 PRINT "BED SLOPE S=";S
310 PRINT "SLOPE OF THE CHANNEL WALLS Z=";Z
320 '
330 '--------  DEFINE FUNCTIONS  -------------------------------------------------
340 DEF FNA(B,D,Z)=(B+Z*D)*D
350 DEF FNR(AREA,B,D,Z)=AREA/(B+2*D*SQR(Z*Z+1))
360 DEF FNQ(N,AREA,HYDR,S)=1.486/N*AREA*HYDR^(2/3)*SQR(S)
361 '
362 '--------  INPUT ESTIMATED DEPTHS  -------------------------------------------
370 PRINT :PRINT :PRINT :PRINT
380 COLOR 4,0,0
390 PRINT "ENTER TWO DEPTHS THAT SPAN THE RANGE OF THE REQUIRED DEPTH."
400 PRINT
410 INPUT "WHAT IS THE MINIMUM DEPTH";MINDEPTH
420 INPUT "WHAT IS THE MAXIMUM DEPTH";MAXDEPTH
430 '
440 '--------  FLOW AT DEFINED DEPTHS  -------------------------------------------
450 AREAMIN=FNA(B,MINDEPTH,Z)
460 RADMIN=FNR(AREAMIN,B,MINDEPTH,Z)
470 QMIN=FNQ(N,AREAMIN,RADMIN,S)
480 AREAMAX=FNA(B,MAXDEPTH,Z)
```

```
490 RADMAX=FNR(AREAMAX,B,MAXDEPTH,Z)
500 QMAX=FNQ(N,AREAMAX,RADMAX,S)
510 '
520 '--------  CHECK FOR OPPOSITE SIGNS  ---------------------------------------
530 IF SGN(QMIN-Q) <> SGN(QMAX-Q) THEN GOTO 630
540 PRINT :PRINT :PRINT :PRINT
550 COLOR 31,4,8
560 PRINT "MINIMUM DEPTH=";MINDEPTH;"YIELDS A DISCHARGE=";QMIN
570 PRINT "MAXIMUM DEPTH=";MAXDEPTH;"YIELDS A DISCHARGE=";QMAX
580 PRINT "THE REQUIRED DISCHARGE=";Q
590 PRINT "THE ESTIMATED DEPTHS ENTERED ARE NOT ACCEPTABLE."
600 PRINT "PLEASE PREPARE NEW ESTIMATES."
610 GOTO 362
620 '
630 '--------  CHECK FOR ESTIMATED DISCHARGE EQUAL TO REQUIRED  ----------------
640 IF QMIN=Q THEN GOTO 660
650 GOTO 690                                          ' EMPTY FALSE BLOCK
660 REQDDEPTH=MINDEPTH
670 REQDQ=QMIN
675 AREAOK=AREAMIN
680 GOTO 1210                                         ' SENDS TO OUTPUT
690 IF QMAX=Q THEN GOTO 710
700 GOTO 740                                          ' EMPTY FALSE BLOCK
710 REQDDEPTH=MAXDEPTH
720 REQDQ=QMAX
725 AREAOK=AREAMAX
730 GOTO 1210                                         ' SENDS TO OUTPUT
740 '
750 '--------  FIRST ITERATION  ------------------------------------------------
760 MIDDEPTH=(MINDEPTH+MAXDEPTH)/2
770 AREAMID=FNA(B,MIDDEPTH,Z)
780 RADMID=FNR(AREAMID,B,MIDDEPTH,Z)
790 QMID=FNQ(N,AREAMID,RADMID,S)
800 '
810 '--------  TEST FOR FIRST ITERATION EQUAL TO REQUIRED FLOW  ----------------
820 IF QMID=Q THEN GOTO 1170                          ' SENDS TO ASSIGNMENT
830 '
840 '--------  REPLACE CORRESPONDING DEPTH WITH MIDDEPTH (FIRST ITERATION)  ----
850 IF SGN(QMID-Q)=SGN(QMIN-Q) THEN GOTO 870
860 GOTO 890                                          ' EMPTY FALSE BLOCK
870 QMIN=QMID
880 MINDEPTH=MIDDEPTH
890 IF SGN(QMID-Q)=SGN(QMAX-Q) THEN GOTO 910
900 GOTO 930                                          ' EMPTY FALSE BLOCK
910 QMAX=QMID
920 MAXDEPTH=MIDDEPTH
930 LASTDEPTH=MIDDEPTH                                ' SETS COMPARISON
940 '
950 '--------  SUBSEQUENT ITERATIONS  ------------------------------------------
960 MIDDEPTH=(MINDEPTH+MAXDEPTH)/2
```

```
970 AREAMID=FNA(B,MIDDEPTH,Z)
980 RADMID=FNR(AREAMID,B,MIDDEPTH,Z)
990 QMID=FNQ(N,AREAMID,RADMID,S)
1000 '
1010 '--------  TEST FOR SUBSEQUENT ITERATIONS EQUAL TO REQUIRED FLOW  ---------
1020 IF QMID=Q THEN GOTO 1170                              ' SENDS TO ASSIGNMENT
1030 '
1040 '--------  REPLACE CORRESPONDING DEPTH WITH MIDDEPTH  ---------------------
1050 IF SGN(QMID-Q)=SGN(QMIN-Q) THEN GOTO 1070
1060 GOTO 1090                                             ' EMPTY FALSE BLOCK
1070 QMIN=QMID
1080 MINDEPTH=MIDDEPTH
1090 IF SGN(QMID-Q)=SGN(QMAX-Q) THEN GOTO 1110
1100 GOTO 1130                                             ' EMPTY FALSE BLOCK
1110 QMAX=QMID
1120 MAXDEPTH=MIDDEPTH
1130 IF ABS(LASTDEPTH-MIDDEPTH)<.000001 THEN GOTO 1170      ' SENDS TO ASSIGNMENT
1140 LASTDEPTH=MIDDEPTH                                    ' RESET COMPARISON
1150 GOTO 950                                              ' START NEW ITERATION
1160 '
1170 '--------  ASSIGNMENT OF DEPTH, DISCHARGE AND CHANNEL AREA  ---------------
1180 REQDDEPTH=MIDDEPTH
1190 REQDQ=QMID
1195 AREAOK=AREAMID
1200 '
1210 '--------  OUTPUT  ---------------------------------------------------------
1220 PRINT :PRINT :PRINT :PRINT
1230 COLOR 5,0,0
1240 PRINT "FOR A REQUIRED DISCHARGE OF";Q
1250 COLOR 6,0,0
1260 PRINT "THE REQUIRED DEPTH OF THE CHANNEL IS";REQDDEPTH
1270 COLOR 7,0,0
1280 PRINT "THIS YIELDS A CHANNEL CROSS SECTIONAL AREA OF";AREAOK
1290 COLOR 9,0,0
1300 PRINT "THE ACTUAL DISCHARGE SUPPORTED BY THIS AREA IS";REQDQ
1310 COLOR 10,0,0
1320 PRINT "THE PERCENTAGE DIFFERENCE IN DISCHARGE IS";(REQDQ-Q)/Q*100;"%"
1330 '
1340 '--------  USER CONTROL FOR NEW PROBLEM  -----------------------------------
1350 PRINT :PRINT :PRINT :PRINT
1360 COLOR 11,0,0
1370 INPUT "DO YOU HAVE ANOTHER DISCHARGE PROBLEM (Y/N)";ANS$
1380 IF ANS$="N" OR ANS$="n" THEN GOTO 1400
1390 GOTO 160                                               ' RETURN TO INPUT
1400 END
```

```
Program ManningsFlow;

var
   b, n, q, s, z,
   mindepth, maxdepth, middepth,
   areamin, areamax, areamid,
   qmin, qmax, qmid,
   lastdepth:                          real;
   answer:                             char;

function Sgn(FX: real): integer;
begin
   if FX = 0.0
      then Sgn:=0
   else begin
      if abs(FX)=FX
         then Sgn:=1
         else Sgn:=-1
      end;
end;

function A(b,d,z: real): real;
begin
A:=(b+z*d)*d;
end;

function R(fa,b,d,z: real): real;
begin
R:=fa/(b+2*d*sqrt(z*z+1));
end;

function FQ(n,fa,hr,s: real): real;
begin
FQ:=1.486/n*fa*exp((2/3)*ln(hr))*sqrt(s);
end;

begin

clrscr;
writeln('This program uses the binary search (half interval) method');
writeln('to find the depth of a trapezoidal channel required for a given');
writeln('discharge using Mannings equation:  Q=1.486/N*A*R^(2/3)*S^(1/2)');
writeln('Where:  Q=discharge, N=Mannings coefficient, S=bed slope,');
writeln('        Z=slope of channel walls, B=minimum width of channel,');
writeln('        D=depth of channel A=area of channel  =(B+Z*D)*D, and');
writeln('        R=hydraulic radius =A/(B+2*D*Sqrt(Sqr(Z)+1)).');
writeln('Later you will be asked for a range you believe D will appear in.');

repeat {until answer is n or N.  No more problems.}
```

```
writeln; writeln; writeln;
write('What is Mannings coefficient, N ?  ');
readln(n);
write('What is the minimum width of the channel, B ? ');
readln(b);
write('What is your required discharge, Q ? ');
readln(q);
write('What is the bed slope, S ? ');
readln(s);
write('What is the slope of the channel walls, Z ? ');
readln(z);
writeln; writeln;
writeln('You have defined the following:');
writeln('N= ',n,' B= ',b,' Q= ',q);
writeln('S= ',s,' Z= ',Z);

repeat  (until Sgn(qmin-q) <> Sng(qmax-q))
   writeln; writeln;
   writeln('You now need to enter your estimates of the depth as asked:');
   write('What is the minimum depth you expect? ');
   readln(mindepth);
   write('What is the maximum depth you expect? ');
   readln(maxdepth);
   writeln; writeln;
   writeln('The minimum depth you expect is ',mindepth);
   writeln('The maximum depth you expect is ',maxdepth);

   areamin:=A(b,mindepth,z);   {call function A}
   qmin:=FQ(n,areamin,R(areamin,b,mindepth,z),s);  {calls functions FQ & R}
   areamax:=A(b,maxdepth,z);   {call function A}
   qmax:=FQ(n,areamax,R(areamax,b,maxdepth,z),s);  {calls functions FQ & R}

   if Sgn(qmin-q)=Sgn(qmax-q) then begin
      writeln; writeln;
      writeln('Min depth=',mindepth,' therefore min flow=',qmin);
      writeln('Max depth=',maxdepth,' therefore max flow=',qmax);
      writeln('These depths are not acceptable.  Please try two new depths.')
   end;

until Sgn(qmin-q) <> Sgn(qmax-q);

if qmin = q then begin
   middepth:=mindepth;
   areamid:=areamin;
   qmid:=qmin
end
else begin
   if qmax = q then begin
      middepth:=maxdepth;
      areamid:=areamax;
```

```
        qmid:=qmax
      end
      else begin
         middepth:=(mindepth+maxdepth)/2;
         areamid:=A(b,middepth,z);   {call function A}
         qmid:=FQ(n,areamid,R(areamid,b,middepth,z),s);  {calls functions FQ & R}
         if qmid <> q then begin
            if Sgn(qmid-q)=Sgn(qmin-q) then begin
               mindepth:=middepth;
               areamin:=areamid;
               qmin:=qmid
            end
            else begin
               maxdepth:=middepth;
               areamax:=areamid;
               qmax:=qmid
            end;
            repeat {until abs(lastdepth-middepth)<1E-06 or qmid=0.0}
               lastdepth:=middepth;
               middepth:=(mindepth+maxdepth)/2;
               areamid:=A(b,middepth,z);
               qmid:=FQ(n,areamid,R(areamid,b,middepth,z),s);
               if qmid <> q then begin
                  if Sgn(qmid-q)=Sgn(qmin-q) then begin
                     mindepth:=middepth;
                     areamin:=areamid;
                     qmin:=qmid
                  end
                  else begin
                     maxdepth:=middepth;
                     areamax:=areamid;
                     qmax:=qmid
                  end;
               end;
            until (abs(lastdepth-middepth) < 1E-06) or (qmid=q)
         end
      end
   end;
   writeln; writeln; writeln;
   writeln('The depth required to allow a discharge of ',q);
   writeln('is ',middepth,'.');
   writeln('This depth yields an area of ',areamid);
   writeln('The actual discharge will be ',qmid);
   writeln('The resulting per centage error is ',(qmid-q)/q*100);
   writeln; writeln;
   write('Is there another problem to be solved (Y/N)? ');
   readln(answer)
until (answer='N') or (answer='n')  {no remaining problems}
end.
```

```
10 '-------------------------------------------------------------------------
20 '         PROGRAM ASNGRADE ASSIGNS LETTER GRADES TO NUMERICAL SCORES
30 '         INPUT IS FROM DATA STATEMENTS FOUND AT THE END OF LISTING.
40 '         VARIABLE STUNAME HOLDS CHARACTER STRING FOR STUDENTS' NAMES.
50 '         VARIABLE SCORE HOLDS THE NUMERICAL SCORE.
60 '         VARIABLE GRADE HOLDS TYHE LETTER GRADE.
70 '         VARIABLE NOSTUDNT IS THE NUMBER OF STUDENTS IN THE CLASS.
80 '-------------------------------------------------------------------------
90 PRINT "THIS PROGRAM ASSIGNS LETTER GRADES TO NUMERICAL SCORES."
100 PRINT "THE DATA IS READ INTO THE COMPUTER VIA DATA STATEMENTS"
110 PRINT "FOUND AT THE END OF THE PROGRAM.  THE NAMES ARE LIMITED"
120 PRINT "TO 15 CHARACTERS.  THE FIRST DATA STATEMENT MUST CONTAIN AN"
130 PRINT "INTEGER WHICH IS THE NUMBER OF STUDENTS' GRADES BEING SOUGHT."
140 PRINT : PRINT
150 PRINT TAB(11);"NAME"; TAB(24);"SCORE"; TAB(32);"GRADE"
160 PRINT TAB(5);"---------------"; TAB(24);"-----"; TAB(32);"-----"
170 READ NOSTUDNT%                        'READS NO. OF STUDENTS IN THE CLASS
180 FOR I=1 TO NOSTUDNT%
190     READ STUNAME$, SCORE                          'INPUTS NAME AND SCORE
200     IF SCORE >= 90! THEN GRADE$="A"                    'ASSIGNMENT
210     IF SCORE < 90! AND SCORE >=80! THEN GRADE$="B"      '    OF
220     IF SCORE < 80! AND SCORE >=70! THEN GRADE$="C"      '   THE
230     IF SCORE < 70! AND SCORE >=60! THEN GRADE$="D"      '  GRADE
240     IF SCORE < 60! THEN GRADE$="F"                      ' SECTION
250     PRINT  TAB(5)STUNAME$; TAB(24)SCORE; TAB(34)GRADE$  'OUTPUT LINE
260     IF SCORE>100 OR SCORE<0 THEN PRINT "LAST SCORE OUT OF USUAL RANGE!!!"
270 NEXT I
280 PRINT : PRINT
290 PRINT "*****  ALL GRADES ASSIGNED!  *****"  : PRINT
300 DATA 11
310 DATA "PAUL BALSERAK", 70
320 DATA "WILLIAM CADDEN", 80
330 DATA "GREGORY CURREY", 90
340 DATA "CARL DUNCAN", 99
350 DATA "LISBETH FREEMAN", 95
360 DATA "TODD GREENWOOD", 85
370 DATA "DAVID HALL", 55
380 DATA "ROBERT HUNTGATE",66
390 DATA "PAUL KNAEBEL", 78
400 DATA "KEVIN LONG", 88
410 DATA "HELEN STEYER", 91
420 END
```

```
Program AssignGrade;

var
   i, nostudents:       integer;
   score:               real;
   stuname, grade:      string[15];

begin

clrscr;
writeln('This program assigns letter grades to numeric scores input');
writeln('from the keyboard.');
writeln('NAME holds character string for students names.');
writeln('SCORE holds the numeric score.');
writeln('GRADE holds the letter grade.');
writeln('NOSTUDENTS is the number of students grades to be processed.');
writeln; writeln;
write('What is the number of grades to be processed?  ');
readln(nostudents);
for i:=1 to nostudents do begin
   writeln; writeln;
   write('What is the students name?  ');
   readln(stuname);
   write('What is this students score? ');
   readln(score);
  if score>=90
     then grade:='A'
    else begin
     if score>=80
       then grade:='B'
      else begin
       if score>=70
          then grade:='C'
         else begin
          if score>=60
             then grade:='D'
             else grade:='F'
          end}
       end;
     end;
  end;
  writeln; writeln;
  writeln(stuname:15,' has a numeric score of',score,' and a grade of ',grade);
  if (score>100) or (score<0) then begin
     writeln; writeln;
     writeln('Score is outside usual range, please check score.')
  end;
end;
end.
```

CE 4980 Problem Set 4 Due: Feb. 12, 86

1. NUMERICAL INTEGRATION

Write a program to evaluate the integral

$$I = \int_a^b f(x)\,dx$$

using Simpson's rule. Input to the program shall consist of the upper and lower limits of the integral XU and XL and the desired tolerance EPSILON. You should begin by evaluating the integral using two intervals, then double the number of intervals after each step until the desired accuracy is achieved.

Use the following statement to test for convergence

```
IF (A.CURR - A.PREV) < ABS(A.CURR * EPSILON) THEN (quit)
```

where A.CURR is the current value of the integral, and A.PREV the value of the integral from the previous step. Note that the statement uses a relative tolerance criterion to avoid problems which may occur if the value of the integral is small (e.g., 1E-12).

Test your program by comparing your results with those for several analytical solutions.

Your program should print the number of intervals and the current value of the integral after each step.

CE 4980 Problem Set 4

2. The axial load capacity of a steel column is:

$$Pa = Fa\ A$$

where Fa is the allowable stress and A the area of the column. The allowable stress as given by the American Institute for Steel Construction is:

$$Fa = \begin{cases} \dfrac{Fy\left[1 - (kl/r)^2/(2Cc^2)\right]}{FS} & \text{for } kl/r \le Cc \\ 12\pi^2 E/\left[23(kl/r)^2\right] & \text{for } kl/r > Cc \end{cases}$$

where FS = factor of safety

$$= \frac{5}{3} + \frac{3(kl/r)}{8\ Cc} - \frac{1(kl/r)^3}{8\ Cc}$$

$$Cc = \sqrt{2\pi^2 E/Fy}$$

(kl/r) = slenderness ratio

r = radius of gyration

l = length of column

E = modulus of elasticity (29,000 ksi)

Fy = yield stress of steel

Note that the above equations only apply to main members for which kl/r is less than 200. Also since a column can fail by buckling about either the strong or weak axis, the value of kl/r to be used in the above equations is the larger of $(kl/r)_x$ and $(kl/r)_y$ where the subscripts x and y represent the strong and weak axis.

Write a program to read in Lx, Ly, kx, ky, rx, ry, A, and Fy for a column and compute the allowable load. Your program should print an error message if (kl/r)max is greater than 200.

```
10 CLS
20 PRINT "THIS PROGRAM PERFORMS NUMERICAL INTEGRATION USING SIMPSON'S RULE."
30 PRINT "THE FUNCTION TO BE INTEGRATED IS DEFINED IN PROGRAM LINE 180."
40 PRINT "IT WILL NEED TO BE CHANGED BEFORE CONTINUING."
50 INPUT "HAVE YOU CHANGED THE FUNCTION TO BE INTEGRATED";ANS$
60 IF ANS$="N" OR ANS$="n" THEN GOTO 610          'GO TO END
69 '
70 '--------  INPUT LIMITS OF INTEGRATION AND REQUIRED TOLERANCE  -------------
80 PRINT :PRINT :
90 INPUT "WHAT IS THE LOWER LIMIT OF INTEGRATION";XL
100 INPUT "WHAT IS THE UPPER LIMIT OF INTEGRATION";XU
110 INPUT "WHAT IS THE REQUIRED TOLERANCE";EPSILON
119 '
120 '--------  ECHO INPUT  -----------------------------------------------------
130 PRINT :PRINT
140 PRINT "THE FUNCTION YOU HAVE DEFINED IN PROGRAM LINE 180 WILL BE INTEGRATED"
150 PRINT "FROM";XL;" TO";XU;"."
160 PRINT "YOU HAVE A REQUIRED TOLERANCE OF";EPSILON;"."
169 '
170 '--------  DEFINE THE FUNCTION TO BE INTEGRATED  ---------------------------
180 DEF FNF(X)=X*SIN(2*X)
189 '
190 '--------  INITIALIZE NO. OF INTERVALS & STEP COUNTER  ----------------------
200 NOINT%=2
210 ITSTEP%=1
219 '
220 '--------  SET UP & PRINT HEADINGS  ----------------------------------------
230 PRINT :PRINT
240 PRINT TAB(5)"STEP";TAB(13)"NO. OF INTERVALS";TAB(33)"CURRENT VALUE OF THE INTEGRAL"
250 PRINT TAB(5)"----";TAB(13)"---------------";TAB(33)"----------------------------"
259 '
260 '--------  FIRST ITERATION  ------------------------------------------------
270 H=(XU-XL)/NOINT%
280 FXL=FNF(XL)
290 FXU=FNF(XU)
300 XM=XL+H
310 AREA=H*(FXL+FXU+4*FNF(XM))/3
320 PRINT TAB(5)ITSTEP%;TAB(18)NOINT%;TAB(44)AREA
330 '
340 '--------  SUBSEQUENT ITERATIONS  ------------------------------------------
350 '
360 '  INCREMENT NOINT%, STEP%, AREA.PREV, & H
370 NOINT%=2*NOINT%
380 ITSTEP%=ITSTEP%+1
390 AREA.PREV=AREA
400 H=(XU-XL)/NOINT%
409 '
410 '  SET UP ODD SUM
420 FOR I=1 TO NOINT%-1 STEP 2
430 XI=XL+H*I
```

```
440 ODDSUM=ODDSUM+FNF(XI)
450 NEXT I
459 '
460 '  SET UP EVEN SUM
470 FOR I=2 TO NOINTX-2 STEP 2
480 XI=XL+H*I
490 EVENSUM=EVENSUM+FNF(XI)
500 NEXT I
509 '
510 '  CALCULATE AREA
520 AREA=H*(FXU+FXL+4*ODDSUM+2*EVENSUM)/3
529 '
530 '  PRINT ANSWERS FOR CURRENT STEP
540 PRINT TAB(5)ITSTEPX;TAB(18)NOINTX;TAB(44)AREA
549 '
550 '  TEST FOR CONVERGENCE
560 IF ABS(AREA-AREA.PREV) > ABS(AREA*EPSILON) THEN GOTO 564   'RESET SUMS
562 GOTO 570              '********** CONVERGENCE REACHED **********
564 ODDSUM=0              'RESET SUMS
566 EVENSUM=0             '  "    "
568 GOTO 340              'START NEW ITERATION
569 '
570 '  CONVERGENCE REACHED
571 '
580 '--------  PRINT RESULTS  ------------------------------------------------
590 PRINT :PRINT :PRINT :PRINT
600 PRINT "THE VALUE OF YOUR INTEGRAL IS ";AREA;"."
610 PRINT :PRINT :PRINT :PRINT :END
```

```
program NumericalIntegration;   (using Simpson's Rule)

   var  xl,xu,tol:real;


  function F(x:real):real;   (the function to be integrated)
     begin   F:=x*sin(2*x)       end;


  function dx(range:real; sub:integer):real;   (length of interval)
     begin   dx:=range/sub      end;


  function Area(xl,xu,range,odd,even:real; sub:integer):real;  {Area under curve}
     begin   Area:=(F(xl)+F(xu)+4*odd+2*even)*dx(range,sub)/3     end;
```

```
function Sum(K,sub:integer; xl,H:real):real;   (performs summations)
   var    Tsum:real;   I:integer;
   begin   Tsum:=0;   I:=K;         (initialize variables)
           while I <= (sub-K)  do begin
              Tsum:=Tsum+F(xl+I*H);
              I:=I+2    end;  (of while)
   Sum:=Tsum            end;  (of function Sum)


procedure Data(var xl,xu,tol:real);
   begin
      clrscr;  writeln('***  Remember to change F(x)!  ***');
      writeln;  writeln('----  Integration by Simpsons Rule  ----');
      writeln; writeln;
      write('xl= ');readln(xl);
      write('xu= ');readln(xu);
      write('tol= ');readln(tol)
   end;   (of Data)


procedure Echo(xl,xu,tol:real);
   begin
      writeln; writeln; writeln;
      writeln('xl=',xl:8:3,'     xu=',xu:8:3,'     tol=',tol:7)
   end;  (of Echo)


procedure Headings;
   begin
      writeln; writeln;
      writeln('    Step    Intervals    Current Area');
      writeln('    ----    ---------    ------------')
   end;  (of Headings)


procedure Busywork(var xl,xu,tol:real);
   begin
      Data(xl,xu,tol);
      Echo(xl,xu,tol);
      Headings
   end;  (of Busywork)


procedure Output(step,sub:integer; A:real);
   begin
      writeln(' ':4,step:4,' ':6,sub:4,' ':7,A:12:6)
   end;  (of Output)
```

```
procedure Increment(var lastA,A:real; var step,sub:integer);
   begin
      lastA:=A;
      step:=step+1;
      sub:=2*sub
   end;  {of Increment}


procedure Initial(var range,xl,xu,A:real; var step,sub:integer);
   begin
      range:=xu-xl;
      step:=1;
      sub:=2;
      A:=(F(xl)+F(xu)+4*F(xl+dx(range,sub)))*dx(range,sub)/3;
      Output(step,sub,A)
   end;  {of Initial}


procedure Subsequent(var A,range,xl,xu,tol:real; var step,sub:integer);
   var   lastA   :real;
   begin
      repeat
         Increment(lastA,A,step,sub);
         A:=Area(xl,xu,range,sum(1,sub,xl,dx(range,sub)),sum(2,sub,xl,dx(range,sub)),sub);
         Output(step,sub,A)
      until abs(A-lastA) < abs(A*tol);
   end;  {of Subsequent}


procedure Final(A:real);
   begin
      writeln; writeln;
      writeln('The value of your integral is ',A)
   end;


procedure Calculations(var xl,xu,tol:real);
   var
      range, A            :real;
      step, sub           :integer;
   begin
      Initial(range,xl,xu,A,step,sub);
      Subsequent(A,range,xl,xu,tol,step,sub);
      Final(A)
   end;  {of Calculations}

begin
   Busywork(xl,xu,tol);
   Calculations(xl,xu,tol)
end.  {of Main Program}
```

```
10 CLS
20 PRINT "THIS PROGRAM WILL COMPUTE THE ALLOWABLE LOAD FOR A STEEL COLUMN AS"
30 PRINT "GIVEN BY AISC EQUATION (1.5-1) OR (1.5-2)."
40 PRINT "THE REQUIRED INPUT DATA IS THE LENGTH, EFFECTIVE LENGTH FACTOR, AND"
50 PRINT "RADIUS OF GYRATION ABOUT BOTH THE X AND Y AXES, THE GROSS AREA OF"
60 PRINT "THE COLUMN, AND THE YIELD STRESS OF THE STEEL BEING USED."
70 PRINT :PRINT
80 PRINT "THE AISC RECOMMENDS THE FOLLOWING EFFECTIVE LENGTH FACTORS:"
90 PRINT "FIXED-FIXED                                    0.65"
100 PRINT "FIXED-PINNED                                   0.80"
110 PRINT "FIXED-TRANSLATION FREE & ROTATION FIXED        1.20"
120 PRINT "PINNED-PINNED                                  1.00"
130 PRINT "FIXED-FREE                                     2.10"
140 PRINT "PINNED-TRANSLATION FREE & ROTATION FIXED       2.00"
150 '
160 '--------  INPUT  ------------------------------------------------------
170 PRINT :PRINT :PRINT :PRINT
180 INPUT "WHAT IS THE YIELD STRESS OF YOUR STEEL IN KSI";FY
190 INPUT "WHAT IS THE GROSS AREA OF YOU COLUMN IN INCHES SQUARED";A
200 PRINT "----- X AXIS PROPERTIES -----"
210 INPUT "WHAT IS THE LENGTH OF YOUR COLUMN IN DECIMAL FEET";LX
220 INPUT "WHAT IS THE EFFECTIVE LENGTH FACTOR OF YOUR COLUMN";KX
230 INPUT "WHAT IS THE RADIUS OF GYRATION OF YOUR COLUMN IN INCHES";RX
240 PRINT "----- Y AXIS PROPERTIES -----"
250 INPUT "WHAT IS THE LENGTH OF YOUR COLUMN IN DECIMAL FEET";LY
260 INPUT "WHAT IS THE EFFECTIVE LENGTH FACTOR OF YOUR COLUMN";KY
270 INPUT "WHAT IS THE RADIUS OF GYRATION OF YOUR COLUMN IN INCHES";RY
280 '
290 '--------  ECHO INPUT  -------------------------------------------------
300 PRINT :PRINT :PRINT :PRINT
310 PRINT "FY=";FY;"KSI    A=";A;"IN^2"
320 PRINT "LX=";LX;"FT    KX=";KX;"    RX=";RX;"IN"
330 PRINT "LY=";LY;"FT    KY=";KY;"    RY=";RY;"IN"
340 '
350 '--------  DEFINE CONSTANTS  -------------------------------------------
360 PI=3.141592653594
370 E=29000    'KSI = MODULUS OF ELASTICITY
380 PRINT "THE MODULUS OF ELASTICITY IS TAKEN AS";E;"KSI."
390 '
400 '--------  CALCULATE CC  -----------------------------------------------
410 CC=SQR(2*PI*PI*E/FY)
420 '
430 '--------  SLENDERNESS RATIO  ------------------------------------------
440 'X AXIS
450 SRX=KX*LX*12/RX
460 'Y AXIS
470 SRY=KY*LY*12/RY
480 'CONTROLING
490 IF SRX > SRY THEN SR=SRX:  KL=SR*RX:  AXIS$="X":  GOTO 510  'X AXIS CONTROLS
500 SR=SRY:  KL=SR*RY:  AXIS$="Y"                               'Y AXIS CONTROLS
```

```
510 '
520 '--------  CHECK FOR SR > 200  ------------------------------------------
530 IF SR < 200 THEN GOTO 574             'SR < MAX ALLOWED   OK
540 PRINT :PRINT
550 PRINT "ERROR:  CONTROLING SLENDERNESS RATIO =";SR;" > SRMAX=200 BY THE"
560 PRINT "        AISC SPEC 1.8.4.  THE CONTROLING AXIS IS THE ";AXIS$;"AXIS."
561 PRINT "        THE EFFECTIVE LENGTH ABOUT THIS AXIS IS";KL;"IN."
562 PRINT "        RECHECK YOUR LENGTHS AND EFFECTIVE LENGTH COEFFICIENTS."
570 GOTO 720                              'GOTO END
574 PRINT :PRINT
575 PRINT "THE CONTROLING SLENDERNESS RATIO IS";SR;".  IT IS ABOUT"
576 PRINT "THE ";AXIS$;" AXIS WHICH HAS AN EFFECTIVE LENGTH OF";KL;"IN."
577 PRINT "THIS EFFECTIVE LENGTH EQUALS";KL/12;"FT."
580 '
590 '--------  CALCULATE ALLOWABLE LOAD  -------------------------------------
600 'TEST FOR APPROPRTIATE EQUATION
610 IF SR > CC THEN GOTO 650
620 'EQUATION (1.5-1):
630 PA=A*FY*(1-SR*SR/(2*CC*CC))/(5/3+3*SR/(8*CC)-SR^3/(8*CC^3))
640 GOTO 680                              'GOTO PRINTOUT
650 'EQUATION (1.5-2):
660 PA=A*12*PI*PI*E/(23*SR*SR)
670 '
680 '--------  PRINTOUT  ----------------------------------------------------
690 PRINT :PRINT
700 PRINT "THE ALLOWABLE AXIAL LOAD FOR YOUR COLUMN IS";PA;"KIPS."
710 PRINT :PRINT
720 END
```

```
program ColumnAllowableLoad;   {using AISC Eqns. 1.5-1 and 1.5-2}

const   E=29000;{ksi}

var     A,Fy,Kx,Ky,lx,ly,rx,ry:real;


procedure Definition;
  begin
     clrscr;
     writeln('---  Allowable Column Loads by AISC Spec.  ---');
     writeln('Required Input:  A=Area of column,');
     writeln(' ':17,'Fy=Allowable yield stress in ksi,');
     writeln(' ':17,'K=Effective length factor,');
     writeln(' ':17,'l=Length between bracing in feet,');
     writeln(' ':17,'r=Radius of gyration in inches.');
     writeln;
```

```
      writeln('        Table C1.8.1 from AISC p.5-124');
      writeln('  fixed-fixed                                    0.65');
      writeln('  fixed-pinned                                   0.80');
      writeln('  rotation fixed/translation free-fixed          1.20');
      writeln('  pinned-pinned                                  1.00');
      writeln('  fixed-free                                     2.10');
      writeln('  rotation fixed/translation free-pinned         2.00');
   end;  {of Definition}


procedure Data(var A,Fy,Kx,Ky,lx,ly,rx,ry:real);
   begin
      writeln;writeln;
      write('A= ');readln(A);
      write('Fy= ');readln(Fy);
      writeln('***  X Axis Properties  ***');
      write('Kx= ');readln(Kx);
      write('lx= ');readln(lx);
      write('rx= ');readln(rx);
      writeln('***  Y Axis Properties  ***');
      write('Ky= ');readln(Ky);
      write('ly= ');readln(ly);
      write('ry= ');readln(ry)
   end;  {of Data}


procedure Echo(A,E,Fy,Kx,Ky,lx,ly,rx,ry:real);
   begin
      clrscr;
      writeln('Echo Properties');
      writeln('General:  A=',A:6:2,'sq in, E=',E:6:0,'ksi, Fy=',Fy:4:0,'ksi.');
      writeln('X Axis:   Kx=',Kx:4:2,', lx=',lx:6:3,'ft, rx=',rx:6:3,'in.');
      writeln('Y Axis:   Ky=',Ky:4:2,', ly=',ly:6:3,'ft, ry=',ry:6:3,'in.')
   end;  {of Echo}


procedure Busywork(var A,Fy,Kx,Ky,lx,ly,rx,ry:real);
   begin
      Definition;
      Data(A,Fy,Kx,Ky,lx,ly,rx,ry);
      Echo(A,E,Fy,Kx,Ky,lx,ly,rx,ry)
   end;  {of Busywork}


function ErrorCheck(SR:real):Boolean;
   begin
      if SR > 200
         then  ErrorCheck:=True
         else  ErrorCheck:=False
   end;  {of ErrorCheck}
```

```
function SeparatingPoint(E,Fy:real):real;
   begin    SeparatingPoint:=sqrt(2*Pi*Pi*E/Fy)   end;  (of SeparatingPoint)


procedure SlendernessRatio(var Kx,Ky,lx,ly,rx,ry,SR:real; var Axis:char);
   begin
      if (Kx*lx/rx) >= (Ky*ly/ry)
      then begin
         SR:=Kx*lx*12/rx;
         Axis:='X';
      end
      else begin
         SR:=Ky*ly*12/ry;
         Axis:='Y'
      end; (if)
   end;  (of SlendernessRatio)


procedure BackgroundCalcs(var CC,Fy,Kx,Ky,lx,ly,rx,ry,SR:real; var Axis:char; var Error:Boolean);
   begin
      CC:=SeparatingPoint(E,Fy);
      SlendernessRatio(Kx,Ky,lx,ly,rx,ry,SR,Axis);
      Error:=ErrorCheck(SR)
   end;  (of BackgroundCalcs)


function ShortColumn(Fy,GR:real):real;
   begin
      ShortColumn:=Fy*(1-GR*GR/2)/(5/3+GR*(3-GR*GR)/8)
   end;  (of ShortColumn)


function LongColumn(E,SR:real):real;
   begin
      LongColumn:=12*Pi*Pi*E/(23*SR*SR)
   end;  (of LongColumn)


procedure Output(PA,SR:real; Axis:char);
   begin
      writeln;writeln;writeln;
      writeln('The ',Axis:2,' Axis controls.');
      writeln('The Slenderness Ratio = ',SR:9:5);
      writeln('The maximum allowable axial load = ',PA:7:2)
   end;  (of Output)
```

```
procedure Load(A,CC,E,Fy,SR:real; Axis:char);
   var    PA:real;
   begin
      if SR <= CC
         then   PA:=A*ShortColumn(Fy,(SR/CC))
         else   PA:=A*LongColumn(E,SR);
      Output(PA,SR,Axis)
   end;  {of Load}


procedure ErrorMessage(SR:real; Axis:char);
   begin
      writeln;writeln;writeln;writeln;
      writeln('****  ERROR  ****');
      writeln('SR=',SR:6:2,' about the ',Axis:1,' Axis.');
      writeln('AISC allows a max SR = 200.');
      writeln('Please check your input data.')
   end;  {of ErrorMessage}


procedure Calculations(A,Fy,Kx,Ky,lx,ly,rx,ry:real);
   var     CC,SR:real;   Axis:char;   Error:Boolean;
   begin
      BackgroundCalcs(CC,Fy,Kx,Ky,lx,ly,rx,ry,SR,Axis,Error);
      if Error
         then  ErrorMessage(SR,Axis)
         else  Load(A,CC,E,Fy,SR,Axis)
   end;  {of Calculations}


begin
   Busywork(A,Fy,Kx,Ky,lx,ly,rx,ry);
   Calculations(A,Fy,Kx,Ky,lx,ly,rx,ry)
end.  {of program ColumnAllowableLoad}
```

CE 4980 Problem Set 5 Due: Feb. 21, 1986

LINEAR REGRESSION ANALYSIS

Engineers frequently perform experiments which yield measurements on two variables X and Y. They then attempt to determine the fundamental relationships between these two variables. The most common model is based on the assumption of a linear relationship between X and Y of the form

$$Y = a + bX$$

where a is the intercept, and b the slope of the line passing through the data points. The values of a and b are determined so that the straight line passes through the data points with the least error.

The most widely used technique for fitting a line through a series of observed data points is the least squares method. The least squares mehtod minimizes the squares of the error between the observed data points and the line. The error associated with each point is

$$e_i = y_i - \hat{y}_i = y_i - (a + bx_i)$$

where e_i the ith error is the difference between the observed value $\hat{y}_i$ is the ordinate of the fitting straight line at x_i (Fig. 1).

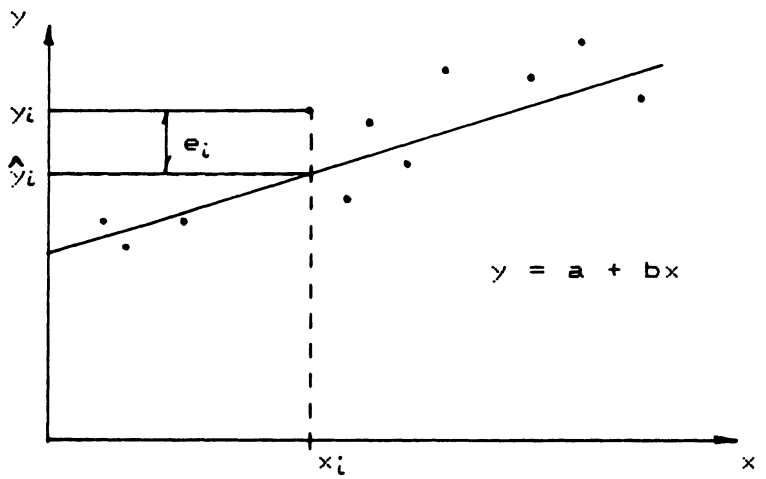


Fig. 1 Regression line and error associated with a point (x_i, y_i).

The sum of the squared errors is

$$S = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} [y_i - (a + bx_i)]^2$$

This sum can be minimized with respect to a and b by taking the partial derivatives of S with respect to a and b and setting the resulting equation equal to zero

$$\frac{\partial S}{\partial a} = -2 \sum_{i=1}^{n} [y_i - (a + bx_i)] = 0$$

$$\frac{\partial S}{\partial b} = -2 \sum_{i=1}^{n} x_i [y_i - (a + bx_i)] = 0$$

The solution of the above simultaneous equations gives

$$b = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{\sum x_i^2 - n\bar{x}^2}$$

$$a = \bar{y} - b\bar{x}$$

where $\bar{x} = \frac{1}{n} \sum x_i$ and $\bar{y} = \frac{1}{n} \sum y_i$

There are several indicators of how well the data can be represented by a straight line. One indicator is the correlation coefficient which can be calculated from

$$r = \frac{\sum x_i y_i - (1/n) \sum x_i \sum y_i}{[\sum x_i^2 - (1/n)(\sum x_i)^2][\sum y_i^2 - (1/n)(\sum y_i)^2]}$$

The value of r ranges from -1 to 1. The correlation coefficient indicates the strength of the linear relationship between X and Y. A value of r close to +1 or -1 indicates that there is a strong linear relationship between X and Y. A value of r close to 0 indicates that there is no relationship between X and Y or that this relationship is non-linear.

Another useful indicator is the coefficient of determination. The coefficient of determination expressis the proportion of the total change in the dependent variable Y which can be explained by the regression line. Thus

$$r^2 = \frac{\text{explained variation}}{\text{total variation}} = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

It is mathematically equal to the square of the correlation coefficient. The value of r ranges from 0 to 1. The closer r is to one, the better the regression equation fits the data. The sample coefficient of determination is usually computed from the following equation

$$r^2 = \frac{a\sum y_i + b\sum x_i y_i - (1/n)(\sum y_i)^2}{\sum y_i^2 - (1/n)(\sum y_i)^2}$$

One other useful indicator is the standard error of the estimate s . This is given by

$$s_e^2 = \frac{1}{n-2}\sum e_i^2 = \frac{\sum(y_i - \hat{y}_i)^2}{n-2}$$

It measures how close the predicted values y is to the measured value y . A zero value indicates a perfect fit. As the fit gradually becomes worse, the standard error increases. It is easier to use the following expression to compute the standard error of the estimate

$$s_e = \frac{\sum y_i^2 - a\sum y_i - b\sum x_i y_i}{n-2}$$

Write a program to fit a straight line to a set of observations (x1,y1), (x2,y2), . . . ,(xn,yn). Compute the values of a and b for the regression line. To determine how well the data can be represented by a straight line, computer also the correlation coefficient, the coefficient of determination and the standard error of the estimate.

Use the following data. The independent variable x represents the number of data points and the dependent variable the computer processing time (in seconds)

X	Y	X	Y	X	Y	X	Y
105	44	330	143	435	208	97	52
511	214	211	112	275	138	187	103
401	193	332	155	55	34	266	110
622	299	322	131	128	73		

n = 15 a = 7.939797 b = 0.4418759
r = 0.986136 r^2 = 0.9724646 s_x = 12.49375

```
10 CLS: DEFINT I,N
20 PRINT "---  LINEAR REGRESSION ANALYSIS  ---": PRINT
30 PRINT "THIS PROGRAM USES THE LEAST SQUARES METHOD TO DETERMINE"
40 PRINT "THE SLOPE AND INTERCEPT OF THE LINE WHICH BEST FITS YOUR"
50 PRINT "N DATA POINTS.  THE CORRELATION COEFFICIENT, THE COEFFICIENT OF"
60 PRINT "DETERMINATION, AND THE STANDARD ERROR ARE ALSO CALCULATED."
70 '
80 '--------  INPUT  -------------------------------------------------------
90 '
100 'INITIALIZE ALL SUMS TO ZERO SO SEVERAL PROBLEMS MAY BE RUN IN SEQUENCE.
110 XSUM=0: YSUM=0: XYSUM=0: X2SUM=0: Y2SUM=0
120 INPUT "N= ";N
130 FOR I= 1 TO N
140    PRINT " X";I;",Y";I;" = ";: INPUT XI,YI
150    XSUM=XSUM+XI: YSUM=YSUM+YI: XYSUM=XYSUM+XI*YI
160    X2SUM=X2SUM+XI*XI: Y2SUM=Y2SUM+YI*YI
170 NEXT I
180 '
190 '--------  CALCULATIONS  ------------------------------------------------
200 XBAR=XSUM/N: YBAR=YSUM/N
210 'SLOPE:
220 B=(XYSUM-N*XBAR*YBAR)/(X2SUM-N*XBAR*XBAR)
230 'INTERCEPT:
240 A=YBAR-B*XBAR
250 'CORRELATION COEFFICIENT:
260 R=(XYSUM-XSUM*YSUM/N)/SQR((X2SUM-XSUM*XSUM/N)*(Y2SUM-YSUM*YSUM/N))
270 'COEFFICIENT OF DETERMINATION:
280 R2=(A*YSUM+B*XYSUM-YSUM*YSUM/N)/(Y2SUM-YSUM*YSUM/N)
290 'STANDARD ERROR:
300 SE=SQR((Y2SUM-A*YSUM-B*XYSUM)/(N-2))
310 '
320 '--------  OUTPUT  ------------------------------------------------------
330 CLS: COLOR 1,0,8
340 PRINT "*******  RESULTS OF LINEAR REGRESSION ANALYSIS  *******": COLOR 2,0,9
350 PRINT :PRINT N;" DATA POINTS WERE USED IN THE ANALYSIS.": COLOR 3,0,10
360 PRINT :PRINT " Y = ";:COLOR 4,0,10:PRINT A;:COLOR 3,0,10:PRINT " + ";
370 COLOR 4,0,10:PRINT B;:COLOR 3,0,10:PRINT " X."
380 COLOR 5,0,11:PRINT :PRINT "CORRELATION COEFFICIENT = ";R
390 COLOR 6,0,12:PRINT "COEFFICIENT OF DETERMINATION = ";R2
400 COLOR 7,0,13:PRINT "STANDARD ERROR = ";SE
410 END
```

```
program LinearRegressionAnalysis;   { Least Squares Method }

var  n:integer;  xsum,ysum,xysum,x2sum,y2sum:real;


procedure Initialize(var xsum,ysum,xysum,x2sum,y2sum:real);
   begin
      xsum:=0;
      ysum:=0;
      xysum:=0;
      x2sum:=0;
      y2sum:=0
   end;  {of Initialize}


procedure Sums(xi,yi:real; var xsum,ysum,xysum,x2sum,y2sum:real);
   begin
      xsum:=xsum+xi;
      ysum:=ysum+yi;
      xysum:=xysum+xi*yi;
      x2sum:=x2sum+xi*xi;
      y2sum:=y2sum+yi*yi
   end;  {of Sums}


procedure Data(var n:integer; var xsum,ysum,xysum,x2sum,y2sum:real);
   var  I:integer;  xi,yi:real;
   begin
      Initialize(xsum,ysum,xysum,x2sum,y2sum);
      writeln;writeln;
      write('n= ');readln(n);
      for I := 1 to n  do begin
         write(' x',I:1,' y',I:1,' = ');
         readln(xi,yi);
         Sums(xi,yi,xsum,ysum,xysum,x2sum,y2sum)
      end;  {of for...do}
   end;  {of Data}


procedure Definition;
   begin
      clrscr;
      writeln('---  Linear Regression Analysis  ---');
      writeln;
      writeln('This program uses the least squares method to fit a straight');
      writeln('line to n data points.  The input required is:  n, the ');
      writeln('number of data points and xi & yi, the related data points,');
      writeln('where i = 1 to n.')
   end;  {of Definition}
```

```
procedure Busywork(var n:integer; var xsum,ysum,xysum,x2sum,y2sum:real);
   begin
      Definition;
      Data(n,xsum,ysum,xysum,x2sum,y2sum)
   end;  {of Busywork}


function Slope(xysum,xbar,ybar,x2sum:real; n:integer):real;
   begin   Slope:=(xysum-n*xbar*ybar)/(x2sum-n*xbar*xbar)   end;  {of Slope}


function Intercept(ybar,b,xbar:real):real;
   begin   Intercept:=ybar-b*xbar   end;  {of Intercept}


procedure Line(n:integer; xsum,ysum,xysum,x2sum:real; var a,b:real);
   var  xbar,ybar:real;
   begin
      xbar:=xsum/n;
      ybar:=ysum/n;
      b:=Slope(xysum,xbar,ybar,x2sum,n);
      a:=Intercept(ybar,b,xbar)
   end;  {of Line}


function Correlation(xysum,xsum,ysum,x2sum,y2sum:real; n:integer):real;
   begin
Correlation:=(xysum-xsum*ysum/n)/sqrt((x2sum-xsum*xsum/n)*(y2sum-ysum*ysum/n))
   end;  {of Correlation}


function Determination(a,ysum,b,xysum,y2sum:real; n:integer):real;
   begin
      Determination:=(a*ysum+b*xysum-ysum*ysum/n)/(y2sum-ysum*ysum/n)
   end;  {of Determination}


function StandardError(y2sum,a,ysum,b,xysum:real; n:integer):real;
   begin   StandardError:=sqrt((y2sum-a*ysum-b*xysum)/(n-2))   end;


procedure Indicators(n:integer; xsum,ysum,xysum,x2sum,y2sum,a,b:real;
                     var r,r2,se:real);
   begin
      r:=Correlation(xysum,xsum,ysum,x2sum,y2sum,n);
      r2:=Determination(a,ysum,b,xysum,y2sum,n);
      se:=StandardError(y2sum,a,ysum,b,xysum,n)
   end;  {of Indicators}
```

```
procedure Output(a,b,r,r2,se:real; n:integer);
   begin
      clrscr; writeln; writeln;
      writeln('***  Results of Linear Regression Analysis  ***');
      writeln;
      writeln('Input consisted of ',n:1,' data points.');
      writeln;
      writeln('Y= ',a:11:9,' + ',b:11:9,'X.');
      writeln;
      writeln('Correlation Coefficient = ',r:12:9);
      writeln('Coefficient of Determination = ',r2:11:9);
      writeln('Standard Error = ',se:12:8)
   end;  (of Output)


procedure Calculations(n:integer; xsum,ysum,xysum,x2sum,y2sum:real);
   var  a,b,r,r2,se:real;
   begin
      Line(n,xsum,ysum,xysum,x2sum,a,b);
      Indicators(n,xsum,ysum,xysum,x2sum,y2sum,a,b,r,r2,se);
      Output(a,b,r,r2,se,n)
   end;  (of Calculations)


begin
   Busywork(n,xsum,ysum,xysum,x2sum,y2sum);
   Calculations(n,xsum,ysum,xysum,x2sum,y2sum)
end.  (of program LinearRegressionAnalysis)
```

Write a program to generate a simple xy graph. Your program should be capable of:

a) reading a data file called GRAPH.DAT containing the x and y coordinates of the points to be plotted. The first record of the data file contains the number of points N.

b) generating x and y values from a user defined function. The number of points generated will be determined by user input.

Store the x and y values in two arrays named X( ) and Y( ).

Write separate subroutines to:

a) Obtain user input.

b) read the coordinates from the data file.

c) generate x and y values from the user defined function.

d) compute the maximum and minimum nalues of x and y.

e) perform necessary scaling and transformations from world coordinates to screen coordinates.

f) draw the graph in high resolution mode.

```
10 CLS
20 '--------  INTRO  ------------------------------------------------------------
30 PRINT "THIS PROGRAM WILL PLOT A SIMPLE X-Y GRAPH FROM EITHER A USER"
40 PRINT "DEFINED FUNCTION OR FROM A FILE NAMED GRAPH.DAT.  THE DATA FILE"
50 PRINT "MUST CONTAIN THE NUMBER OF POINTS TO BE PLOTTED IN THE FIRST"
60 PRINT "RECORD.  SUBSEQUENT RECORDS MUST CONTAIN AN X VALUE FOLLOWED BY"
70 PRINT "A Y VALUE.  FOR A USER DEFINED FUNCTION THE FUNCTION MUST BE CHANGED"
80 PRINT "PRIOR TO RUNNING THE PROGRAM, THEN YOU WILL BE ASKED FOR THE NUMBER"
90 PRINT "OF POINTS TO BE PLOTTED, N, AND FINALLY YOU WILL BE ASKED FOR XMAX"
100 PRINT "AND XMIN.  THE PROGRAM WILL THEN GENERATE THE X VALUES AND THE"
110 PRINT "Y VALUES."
120 '
130 '--------  USER DEFINED FUNCTION  ---------------------------------------
140 DEF FNF(X)=3*SIN(2.5*X)*SIN(20*X)
150 '
160 '--------  DEFINE INTEGERS  ---------------------------------------------
170 DEFINT I,J,K,L,N
180 '
190 '--------  CONTROL TO FILE OR TO FUNCTION  ------------------------------
199 PRINT :PRINT :PRINT :PRINT
200 PRINT "INPUT:  1 -FOR DATA FILE   OR   2 -FOR USER CONTROLLED INPUT."
210 INPUT "WHICH MODE ARE YOU CHOOSING"; MODE
220 IF MODE <> 1 AND MODE <> 2 THEN PRINT "TRY AGAIN PLEASE!":GOTO 190
230 MODE%=CINT(MODE)
240 ON MODE% GOSUB 1000, 2000           ' 1000=GRAPH FILE   2000=USER INPUT
250 IF MODE% = 1 THEN GOSUB 4000        ' X MIN/MAX ONLY IF GRAPH FILE
260 GOSUB 5000                          ' Y MIN/MAX
270 GOSUB 6000                          ' SCALING & TRANSFORMING FUNCTIONS
280 GOSUB 7000                          ' DRAW THE GRAPH
290 END
300 '
310 '
1000 '--------  READ GRAPH FILE  ---------------------------------------------
1010 '
1020 'GENERATES X(I) & Y(I) FROM FILE GRAPH.DAT
1030 OPEN "B:GRAPH.DAT" FOR INPUT AS #1
1040 INPUT #1,N
1050 DIM X(N),Y(N)
1060 FOR I = 1 TO N
1070      INPUT #1, X(I), Y(I)
1080 NEXT I
1090 CLOSE #1
1100 RETURN
1110 '
1120 '
2000 '--------  USER INPUT  --------------------------------------------------
2010 '
2020 'GET NUMBER OF POINTS
2030 INPUT "HOW MANY POINTS DO YOU WANT TO PLOT TODAY";N
2040 '
```

```
2050 'DIMENSION ARRAYS
2060 DIM X(N), Y(N)
2070 '
2080 'ASK FOR RANGE OF X(I)
2090 INPUT "XMIN= ";XMIN
2100 INPUT "XMAX= ";XMAX
2110 '
2120 GOSUB 3000                         ' GENERATES X AND Y VALUES
2130 RETURN
2140 '
2150 '
3000 '--------  GENERATE X AND Y VALUES  ------------------------------------
3010 '
3020 'CALCULATE INTERVAL LENGTH
3030 DX=(XMAX-XMIN)/(N-1)
3040 '
3050 'FIRST VALUES
3060 X(1)=XMIN
3070 Y(1)=FNF(X(1))
3080 '
3090 'SUBSEQUENT VALUES
3100 FOR J = 2 TO N
3110      X(J)=X(J-1)+DX
3120      Y(J)=FNF(X(J))
3130 NEXT J
3140 '
3150 RETURN
3160 '
3170 '
4000 '--------  X MIN/MAX  ----------------------------------------------------
4010 ' FINDS MINIMUM AND MAXIMUM VALUES OF X(I)
4020 XMIN=X(1)
4030 XMAX=X(1)
4040 FOR I = 2 TO N
4050      IF X(I) < XMIN THEN XMIN=X(I)
4060      IF X(I) > XMAX THEN XMAX=X(I)
4070 NEXT I
4080 RETURN
4090 '
4100 '
5000 '--------  Y MIN/MAX  ----------------------------------------------------
5010 ' FINDS MINIMUM AND MAXIMUM VALUES OF Y(I)
5020 YMIN=Y(1)
5030 YMAX=Y(1)
5040 FOR I = 2 TO N
5050      IF Y(I) < YMIN THEN YMIN=Y(I)
5060      IF Y(I) > YMAX THEN YMAX=Y(I)
5070 NEXT I
5080 RETURN
5090 '
```

```
5100 '
6000 '--------  SCALING AND TRANSFORMING X AND Y COORDINATES  ----------------
6010 'DEFINE FUNCTIONS FOR SCALING AND TRANSFORMING WORLD COORDINATES TO
6020 '     SCREEN COORDINATES.
6030 DEF FNXTRANS(MAX,MIN,Z)=CINT(600*(Z-MIN)/(MAX-MIN)+35)
6040 DEF FNYTRANS(MAX,MIN,Z)=CINT(160*(MAX-Z)/(MAX-MIN)+11)
6050 RETURN
6060 '
6070 '
7000 '--------  DRAW GRAPH  ----------------------------------------------------
7010 CLS:  KEY OFF
7020 SCREEN 2                                ' HIGH RESOLUTION GRAPHICS SCREEN
7030 LOCATE 2,37
7040 IF MODE = 2 THEN PRINT "FUNCTION F(X)"
7050 IF MODE = 1 THEN PRINT "FROM GRAPH.DAT"
7060 LOCATE 24,39: PRINT "ABSCISSAS"
7070 LOCATE 8,1: PRINT "O": LOCATE 9,1: PRINT "R": LOCATE 10,1: PRINT "D"
7080 LOCATE 11,1: PRINT "I": LOCATE 12,1: PRINT "N": LOCATE 13,1: PRINT "A"
7090 LOCATE 14,1: PRINT "T": LOCATE 15,1: PRINT "E": LOCATE 16,1: PRINT "S"
7100 LOCATE 2,1: PRINT YMAX
7110 LOCATE 22,1: PRINT YMIN
7120 LOCATE 23,5: PRINT XMIN
7130 LOCATE 23,73: PRINT XMAX
7140 IF XMIN >= 0 THEN LINE(35,173)-(35,9)
7150 IF XMAX <= 0 THEN LINE(635,173)-(635,9)
7160 IF YMIN >= 0 THEN LINE(33,171)-(640,171)
7170 IF YMAX <= 0 THEN LINE(33,11)-(640,11)
7180 IF XMIN < 0 AND XMAX > 0 THEN II=FNXTRANS(XMAX,XMIN,0):LINE(II,176)-(II,9)
7190 IF YMIN < 0 AND YMAX > 0 THEN JJ=FNYTRANS(YMAX,YMIN,0):LINE(30,JJ)-(640,JJ)
7200 PSET(FNXTRANS(XMAX,XMIN,X(1)),FNYTRANS(YMAX,YMIN,Y(1)))
7210 FOR K = 2 TO N
7220      LINE-(FNXTRANS(XMAX,XMIN,X(K)),FNYTRANS(YMAX,YMIN,Y(K)))
7230 NEXT K
7240 TOEND$=INKEY$
7250 IF TOEND$ = "" THEN 7240
7260 SCREEN 0,1: COLOR 4,0,1: LOCATE 13,27: KEY ON
7270 PRINT "FINISHED GRAPHING POINTS!"
7280 LOCATE 24,1:  RETURN
```

```
program GraphPoints;

type matrix = array[1..600] of real;
var  Caseno:integer;
```

```
function F(x:real):real;
   begin
      F:=0.08727*cos(2/2*x)*cos(20*x)
   end;  (of F(x))



function Xtrans(Xmax,Xmin,XI:real):integer;
   begin
      Xtrans:=Round(600*(XI-Xmin)/(Xmax-Xmin)+35)
   end;  (of Xtrans)



function Ytrans(Ymax,Ymin,YI:real):integer;
   begin
      Ytrans:=Round(160*(Ymax-YI)/(Ymax-Ymin)+11)
   end;  (of Ytrans)



procedure MinMax(N:integer; A:matrix; var Amax,Amin:real);
   var   II:integer;
   begin
      Amax:=A[1];
      Amin:=A[1];
      for II:= 2 to N do begin
         if A[II] > Amax
            then Amax:=A[II]
            else if A[II] < Amin
                    then Amin:=A[II]
      end(for)
   end;  (of MinMax)



procedure Values(N:integer; var X,Y:matrix; Xmax,Xmin:real);
   var  II:integer;
        dx:real;
   begin
      dx:=(Xmax-Xmin)/(N-1);
      X[1]:=Xmin;
      Y[1]:=F(Xmin);
      for II:= 2 to N do begin
         X[II]:=X[II-1]+dx;
         Y[II]:=F(X[II])
      end;(for)
   end;  (of Values)
```

```
procedure Titles;
   begin
      GotoXY(37,1);   write('Function F(x)');
      GotoXY(39,25);  write('Abscissas');
      GotoXY(1,8);    write('O');
      GotoXY(1,9);    write('r');
      GotoXY(1,10);   write('d');
      GotoXY(1,11);   write('i');
      GotoXY(1,12);   write('n');
      GotoXY(1,13);   write('a');
      GotoXY(1,14);   write('t');
      GotoXY(1,15);   write('e');
      GotoXY(1,16);   write('s');
   end;  {of Titles}



procedure Labels(Xmax,Xmin,Ymax,Ymin:real);
   begin
      GotoXY(1,2);    write(Ymax:8:2);
      GotoXY(1,22);   write(Ymin:8:2);
      GotoXY(5,23);   write(Xmin:8:2);
      GotoXY(73,23);  write(Xmax:8:2)
   end;  {of Labels}



procedure Axes(Xmax,Xmin,Ymax,Ymin:real);
   var  II,JJ:integer;
   begin
      if Xmin >= 0
         then draw(35,173,35,9,1);
      if Xmax <= 0
         then draw(635,173,635,9,1);
      if Ymin >= 0
         then draw(67,171,640,171,1);
      if Ymax <= 0
         then draw(67,11,640,11,1);
      if (Xmin < 0) and (Xmax > 0) then begin
         II:=Xtrans(Xmax,Xmin,0.0);
         draw(II,173,II,9,1)
      end;{if}
      if (Ymin < 0) and (Ymax > 0) then begin
         JJ:=Ytrans(Ymax,Ymin,0.0);
         draw(33,JJ,640,JJ,1)
      end;{if}
   end;  {of Axes}
```

```
procedure Points(N:integer; X,Y:matrix; Xmax,Xmin,Ymax,Ymin:real);
   var  K,I1,I2,J1,J2:integer;
   begin
      I1:=Xtrans(Xmax,Xmin,X[1]);
      J1:=Ytrans(Ymax,Ymin,Y[1]);
      for K:= 2 to N do begin
         I2:=Xtrans(Xmax,Xmin,X[K]);
         J2:=Ytrans(Ymax,Ymin,Y[K]);
         draw(I1,J1,I2,J2,1);
         I1:=I2;
         J1:=J2
      end;{for}
   end;  {of Points}



procedure FromFile(var N:integer; var X,Y:matrix; var Xmax,Xmin,Ymax,Ymin:real);
   var  GraphDat   :text;
        I          :integer;
        XI,YI      :real;
   begin
      assign(GraphDat,'GRAPH.DAT');
      reset(GraphDat);
      read(GraphDat,N);
      for I:= 1 to N do begin
         read(GraphDat,XI,YI);
         X[I]:=XI;
         Y[I]:=YI
      end;{for}
      close(GraphDat);
      MinMax(N,X,Xmax,Xmin);
      MinMax(N,Y,Ymax,Ymin)
   end;  {of FromFile}



procedure Equation(var N:integer; var X,Y:matrix; var Xmax,Xmin,Ymax,Ymin:real);
   begin
      writeln;writeln;
      write('What is the number of points you wish to plot? ');   readln(N);
      writeln;
      write('Xmin = ');    readln(Xmin);
      write('Xmax = ');    readln(Xmax);
      Values(N,X,Y,Xmax,Xmin);
      MinMax(N,Y,Ymax,Ymin)
   end;  {of Equation}
```

```
procedure Draw(N:integer; X,Y:matrix; Xmax,Xmin,Ymax,Ymin:real);
   begin
      clrscr;
      HiRes;     HiResColor(4);
      Titles;
      Labels(Xmax,Xmin,Ymax,Ymin);
      Axes(Xmax,Xmin,Ymax,Ymin);
      Points(N,X,Y,Xmax,Xmin,Ymax,Ymin);
      repeat      until KeyPressed;
      TextMode;
   end;  (of Draw)



procedure Intro;
   begin
      writeln('This program will plot a simple X-Y graph.');
      writeln('The program will allow you to read from a data file named');
      writeln('GRAPH.DAT or use a user defined function.  For the data file');
      writeln('N, the number of points plotted is in the first record,');
      writeln('then subsequent records contain X[I] and Y[I].  For user');
      writeln('defined functions N will be the number of points evaluated');
      writeln('and plotted (N is input through the keyboard).  Xmax and');
      writeln('Xmin will be entered defining your range.  X[I] and Y[I]');
      writeln('will be computed and then plotted.');
   end;  (of Intro)



procedure Questions(var Caseno:integer);
   begin
      writeln;writeln;writeln;writeln;
      writeln('Input Choices:     1   - for data file');
      writeln('                   2   - for user defined function');
      writeln; writeln;
      write('Caseno = ');  readln(Caseno);
      writeln;writeln;
      if (Caseno <> 1) and (Caseno <> 2) then begin
         writeln;writeln;
         writeln('*****  ERROR  *****');
         writeln('Your case entry was out of range.');
         writeln('You will need to restart from the beginning!');
         writeln('To do so just strike the R key.')
      end;(if)
      if Caseno = 1  then begin
         writeln; writeln;
         writeln('You have chosen to use input from file GRAPH.DAT')
      end;(if)
      if Caseno = 2 then begin
         writeln;writeln;
```

```
        writeln('You have chosed to use input from function F(x)  (line 9)');
        writeln('If you have not changed this function the graph drawn');
        writeln('will not be as you anticipate.')
      end;{if}
   end;  {of Questions}



procedure Submain(Caseno:integer);
   var     N:integer;
           X,Y:matrix;
           Xmax,Xmin,Ymax,Ymin:real;
   begin
      if Caseno = 1
         then FromFile(N,X,Y,Xmax,Xmin,Ymax,Ymin);
      if Caseno = 2
         then Equation(N,X,Y,Xmax,Xmin,Ymax,Ymin);
      if (Caseno = 1) or (Caseno = 2)
         then Draw(N,X,Y,Xmax,Xmin,Ymax,Ymin);
   end;  {of Submain}



begin
   clrscr;
   Intro;
   Questions(Caseno);
   Submain(Caseno)
end.   {of program GraphPoints}
```

Appendix 2

Computer Listing
for
Program FRAME

```
Program Frame;  (11-4-86)
{$I GRAPH.P}
{*****************************************************************************
 *                            PROGRAM FUNCTION                               *
 *****************************************************************************
     The program performs the matrix displacement analysis (fig. 3.9)
     of plane frames composed of prismatic members, which may have
     distinct geometric and material properties.  It handles multiple
     load conditions.  Each load condition may consist of joint loads
     and combinations of the following member actions:  a concentrated
     load at any point of a member (fig. A.9), a uniformly distributed
     load over any portion of the member (fig. A.10), an axial load
     acting at any point on the member (fig. A.11), a uniformly
     distributed axial acting over the entire member, a linearly varying
     (triangular) distributed load acting over a user defined distance,
     a member imperfection (member is not the right length or is bent),
     and a uniform temperature change (eq. 4.23B with Delta T = 0).
     The frame may have internal hinges.  Joint desplacements, such as
     joint settlements, may be prescribed.  Prescribed joint displacements
     are treated according to the Penalty Method (Bathe, p.71 of Holzer's
     text) as dof at the element level.
}

const  mx=30;  mxneq=90;  title='PLANE FRAME ANALYSIS';
       units='UNITS:  kip, inch, radian, fahrenheit';

type  RP=real;
      Matrix=array[1..mxneq,1..mxneq] of ^RP;
      Loaddisp=array[1..mxneq] of ^RP;
      Elemprops=array[1..mx] of ^RP;
      Disp=array[1..6] of ^RP;
      Force=array[1..6,1..mx] of ^RP;
      Gstiff=array[1..7] of ^RP;
      Jfmatrix=array[1..3,1..mx] of ^RP;
      Gcoords=array[1..2,1..mx] of ^RP;
      Member=array[1..6,1..mx] of integer;
      Actions=array[1..mx] of integer;
      Joint=array[1..3,1..mx] of integer;
      Indmat=array[1..6,1..6] of integer;
      Incidence=array[1..2,1..mx] of integer;
      Jdisp=array[1..3,1..mx] of real;
      Dcoords=array[1..mx] of integer;
      Hdat=array[1..2,1..60] of integer;
      S8=string[8];
      S11=string[11];
      S14=string[14];
```

```
var  fname  :S14;         date   :S8;         ch,dr  :char;
     x      :Gcoords;     h      :Hdat;       FRO    :S11;
     f      :Force;       q      :Loaddisp;
     p      :Jfmatrix;    mcode  :Member;
     ss     :Matrix;      jcode  :Joint;
     minc   :Incidence;   na     :Actions;
     area, zi, emod, cte, eleng, c1, c2    :Elemprops;
     ne, nj, nlc, lc, neq, mbd, nh, i, j   :integer;
     outfile, infile                       :text;



(*****************************************************************************
 *                      UTILITIES:  MAX, MIN, and FG                         *
 *****************************************************************************)
function MAX(a,b:integer):integer;        {  Returns the larger of a or b.  }
begin         if a >= b    then MAX:=a    else MAX:=b;         end;  {of MAX}

function MIN(a,b:integer):integer;        {  Returns the smaller of a or b.  }
begin         if a <= b    then MIN:=a    else MIN:=b;         end;  {of MIN}

function FG(c1i,c2i,flx,fly:real):real;      { Transforms local to global.  }
begin                    FG:=c1i*flx+c2i*fly                    end;  {of FG}


(*****************************************************************************
 *                                  CODES                                    *
 *****************************************************************************)
procedure CODES(var mcode:Member; var jcode:Joint; minc:Incidence;
                ne,nj:integer; var neq:integer);
{
     Generate joint code, jcode, by assigning integers in sequence,
     by columns, to all nonzero elements of jcode from 1 to neq;
     generate the member code, mcode, by transferring via minc columns
     of fcode into columns of mcode.

     Generate jcode:
}
var i,j,k,l:integer;
begin
   neq:=0;
   for j:=1 to nj do begin
      for l:=1 to 3 do begin
         if jcode[l,j] <> 0 then begin
            neq:=neq+1;
            jcode[l,j]:=neq;
         end;{if}
      end;{for}
   end;{for}
{
```

```
      Generate mcode:
}
   for i:=1 to ne do begin
      j:=minc[1,i];
      k:=minc[2,i];
      for l:=1 to 3 do begin
         mcode[l,i]:=jcode[l,j];
         mcode[l+3,i]:=jcode[l,k];
      end;{for}
   end;{for}
end;  {of CODES}


{*************************************************************************
 *                                  MODM                                 *
 *************************************************************************}
procedure MODM(var mcode:Member; ne,mn,me:integer; var jdof:integer);
{
      Modify mcode for hinge in member number, mn, at member end, me.
}
var j,l:integer;
begin
   if me=1
      then jdof:=mcode[3,mn]
      else jdof:=mcode[6,mn];
   for j:=1 to ne do begin
      for l:=1 to 6 do begin
         if mcode[l,j] > jdof
            then  mcode[l,j]:=mcode[l,j]+1;
      end;{for}
   end;{for}
   if me=1
      then mcode[3,mn]:=jdof+1
      else mcode[6,mn]:=jdof+1;
end;  {of MODM}


{*************************************************************************
 *                                  MODJ                                 *
 *************************************************************************}
procedure MODJ(var jcode:Joint; nj,jdof:integer);
{
      Modify jcode for hinge in member number, mn, at member end, me.
}
var j,l:integer;
begin
   for j:=1 to nj do begin
      for l:=1 to 3 do begin
         if jcode[l,j] > jdof
            then jcode[l,j]:=jcode[l,j]+1;
```

```
      end;{for}
   end;{for}
end;  {of MODJ}


{*****************************************************************************
 *                                HINGE                                      *
 *****************************************************************************}
procedure HINGE(var mcode:Member; var jcode:Joint; ne,nj:integer;
              var neq,nh:integer; var h:Hdat);
{
     Modify mcode and jcode to account for degrees of freedom
     introduced by internal hinges.
}
var i,jdof,me,mn:integer;
begin
   readln(infile,nh);
   for i:=1 to 3 do writeln(outfile);
   if nh <> 0 then begin
      writeln(outfile,'    INTERNAL HINGES');
      writeln(outfile,'    Member     End');
      for i:=1 to nh do begin
         neq:=neq+1;
         readln(infile,mn,me);
         writeln(outfile,'      ',mn:2,'        ',me:1);
         MODM(mcode,ne,mn,me,jdof);
         MODJ(jcode,nj,jdof);
         h[1,i]:=mn;
         h[2,i]:=me;
      end;{for}  end
   else
      writeln(outfile,'    Frame has no internal hinges.');
end;  {of HINGE}


{*****************************************************************************
 *                                MBAND                                      *
 *****************************************************************************}
function MBAND(mcode:Member; ne:integer):integer;
{
     Compute the half bandwidth, mband, by eq. 6.2:  in each column of
     mcode, the first and last nonzero integers are the smallest and
     largest nonzero integers, respectively, of that column.  MBAND is
     the maximum difference of the nonzero integers in any column of
     mcode.
}
var i,il,idif,is,l,tmband:integer;
begin
   tmband:=0;
   for i:=1 to ne do begin
```

```
      l:=1;
      while (mcode[l,i]=0) and (l < 6) do
         l:=l+1;
      is:=mcode[l,i];
      l:=6;
      while (mcode[l,i]=0) and (l > 1) do
         l:=l-1;
      il:=mcode[l,i];
      idif:=il-is;
      if idif > mband
         then mband:=idif;
   end;{for}
   mband:=mband;
end;  {of MBAND}


{*****************************************************************************
 *                                  PROP                                     *
 *****************************************************************************}
procedure PROP(var x:Gcoords; var area,zi,emod,cte,eleng,c1,c2:Elemprops;
               minc:Incidence; ne,nj:integer);
{
     Read and echo joint coordinates, x[l,j]; compute for each
     element by eqs. C.21 the length, eleng[i], and the direction
     cosines, c1[i], c2[i]; read for each element the cross sectional
     area, area[i], the moment of inertia about the local Z(3)-axis,
     zi[i], the modulus of elasticity, emod[i], and the coefficient
     of thermal expansion, cte[i]; print element properties.
}
var  i,j,k:integer;  el1,el2,t1,t2,t3,t4,t5:real;
begin
   for i:=1 to 3 do writeln(outfile);
   writeln(outfile,'    JOINT COORDINATES');
   writeln(outfile,'    Joint  Direction-1  Direction-2');
   for j:=1 to nj do begin
      readln(infile,el1,el2);
      x[1,j]^:=el1;
      x[2,j]^:=el2;
      writeln(outfile,'     ',j:3,'      ',el1:8:2,'     ',el2:8:2);
   end;{for}
   for i:=1 to 3 do writeln(outfile);
   writeln(outfile,'    ELEMENT PROPERTIES');
   writeln(outfile,'                                  Moment of       ',
          'Elastic       Thermal');
   writeln(outfile,'    Element          Area       Inertia       ',
          'Modulus   Coefficient    Length');
   for i:=1 to ne do begin
      j:=minc[1,i];
      k:=minc[2,i];
      el1:=x[1,k]^-x[1,j]^;
```

```
      el2:=x[2,k]^-x[2,j]^;
      eleng[i]^:=sqrt(el1*el1+el2*el2);
      c1[i]^:=el1/eleng[i]^;
      c2[i]^:=el2/eleng[i]^;
      readln(infile,t1,t2,t3,t4);
      area[i]^:=t1;
      zi[i]^:=t2;
      emod[i]^:=t3;
      cte[i]^:=t4;
      t5:=eleng[i]^;
      writeln(outfile,'      ',i:3,'    ',t1:12:4,'  ',t2:12:4,'  ',
             t3:12:4,'  ',t4:12:4,'  ',t5:8:3);
  end;{for}
end;  (of PROP)


{*****************************************************************************
 *                                  STRUCT                                   *
 *****************************************************************************}
procedure STRUCT(var x:Gcoords; var area,zi,emod,cte,eleng,c1,c2:Elemprops;
                 var mcode:Member; var jcode:Joint; var minc:Incidence;
                 ne,nj:integer; var neq,mbd,nm:integer; var h:Hdat;
                 fname:S14; date:S8);
{
     Read and echo the member incidences, minc[1,i]; initialize the
     elements of the joint code matrix, jcode, to unity, read the number
     of joint constraints, njc; read and echo for each joint constraint
     the the joint number, jnum, and joint direction, jdir, and store a
     zero in the corresponding location of jcode; call CODES, MBAND, and
     PROP.
}
var i,j,jdir,jnum,l,njc:integer;
begin
   for i:=1 to 3 do writeln(outfile);
   writeln(outfile,'    MEMBER INCIDENCES');
   writeln(outfile,'    Member a-end b-end');
   for i:=1 to ne do begin
      readln(infile,minc[1,i],minc[2,i]);
      writeln(outfile,'      ',i:3,'    ',minc[1,i]:3,'   ',minc[2,i]:3);
   end;{for}
   for j:=1 to nj do begin
      for l:=1 to 3 do
         jcode[l,j]:=1;
   end;{for}
   readln(infile,njc);
   for i:=1 to 3 do writeln(outfile);
   writeln(outfile,'    JOINT CONSTRAINTS');
   writeln(outfile,'    Joint  Direction');
   for i:=1 to njc do begin
      readln(infile,jnum,jdir);
```

```
      writeln(outfile,'     ',jnum:3,'     ',jdir:3);
      jcode[jdir,jnum]:=0;
   end;{for}
   CODES(mcode,jcode,minc,ne,nj,neq);
   HINGE(mcode,jcode,ne,nj,neq,nh,h);
   mbd:=MBAND(mcode,ne);
   PROP(x,area,zi,emod,cte,eleng,c1,c2,minc,ne,nj);

end;  (of STRUCT)


{*****************************************************************************
 *                                JLOAD                                      *
 *****************************************************************************}
procedure JLOAD(var q:Loaddisp; jcode:Joint);
{
     Read the number of joint loads, njl; for njl times read the
     joint number, jnum, the joint direction, jdir, and the applied
     force, force, then print jnum, jdir, and force, and finally
     store force in q.
}
var   jfor:real;   njl,jnum,jdir,i,k:integer;
begin
   readln(infile,njl);
   for i:=1 to 3 do writeln(outfile);
   if njl <> 0 then begin
      writeln(outfile,'    JOINT LOADS');
      writeln(outfile,'    Joint  Direction     Force');
      for i:=1 to njl do begin
        readln(infile,jnum,jdir,jfor);
        writeln(outfile,'     ',jnum:3,'       ',jdir:1,'      ',jfor:10:3);
        k:=jcode[jdir,jnum];
        q[k]^:=jfor;
      end;{for}  end{then}
   else
      writeln(outfile,'    NO JOINT FORCES');
end;  (of JLOAD)


{*****************************************************************************
 *                                ASSEMF                                     *
 *****************************************************************************}
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, by eqs. 2.34, 2.37,
     3.92 - 3.94, and the force transformation of section 2.4.
}
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}


{*****************************************************************************
 *                                  MACT                                     *
 *****************************************************************************}
procedure MACT(var f:Force; var q:Loaddisp; area,zi,emod,cte,eleng,c1,
               c2:Elemprops; mcode:Member; var na:Actions; ne:integer);
{
     Read the number of member actions, nmact; for nmact times read
     the member number, mn, the member action type, mat, the action,
     act, and the two distances, dist1, dist2,; print mn, compute and
     accumulate the fixed-end forces; call ASSEMF.
}
var   i,nmact,mn,mat:integer;   act,dist1,dist2,el,a,so,st,b,p,c,d:real;
begin
  readln(infile,nmact);
  for i:=1 to 3 do writeln(outfile);
  if nmact <> 0 then begin
    writeln(outfile,'    MEMBER ACTIONS');
    writeln(outfile,'    Member Type     Action  Distance-1  Distance-2');
    for i:=1 to nmact do begin
      readln(infile,mn,mat,act,dist1,dist2);
      writeln(outfile,'     ',mn:3,'     ',mat:1,'  ',act:10:3,'  ',
              dist1:8:3,'    ',dist2:8:3);
      na[mn]:=na[mn]+1;
      case mat of
        1: begin                    (** CONCENTRATED VERTICAL LOAD  **)
             el:=eleng[mn]^;
             a:=dist1/el;
             f[2,mn]^:=f[2,mn]^-act*(1+a*a*(2*a-3));
             f[3,mn]^:=f[3,mn]^-act*el*a*(1-a)*(1-a);
```

```
      f[5,mn]^:=f[5,mn]^+act*a*a*(2*a-3);
      f[6,mn]^:=f[6,mn]^+act*el*a*a*(1-a);
    end;{1}
  2: begin            (**  UNIFORMLY DISTRIBUTED VERTICAL LOAD  **)
      el:=eleng[mn]^;
      so:=dist1/el;
      st:=dist2/el;
      a:=st*st+st*so+so*so;
      b:=st*st*st+st*st*so+st*so*so+so*so*so;
      p:=act*el/2;
      f[2,mn]^:=f[2,mn]^-p*(st-so)*(2-2*a+b);
      f[3,mn]^:=f[3,mn]^-p*el*(st-so)*(6*(st+so)-8*a+3*b)/6;
      f[5,mn]^:=f[5,mn]^-p*(st-so)*(2*a-b);
      f[6,mn]^:=f[6,mn]^+p*el*(st-so)*(4*a-3*b)/6;
    end;{2}
  3: begin                        (**  CONCENTRATED AXIAL LOAD  **)
      el:=eleng[mn]^;
      a:=dist1/el;
      f[1,mn]^:=f[1,mn]^-act*(1-a);
      f[4,mn]^:=f[4,mn]^-act*a;
    end;{3}
  4: begin               (**  UNIFORMLY DISTRIBUTED AXIAL LOAD  **)
      el:=eleng[mn]^;
      p:=act*el/2;
      f[1,mn]^:=f[1,mn]^-p;
      f[4,mn]^:=f[4,mn]^-p;
    end;{4}
  5: begin         (**  TRIANGULARLY DISTRIBUTED VERTICAL LOAD  **)
      el:=eleng[mn]^;
      so:=dist1/el;
      st:=dist2/el;
      c:=3*st*st+2*st*so+so*so;
      d:=4*st*st*st+3*st*st*so+2*st*so*so+so*so*so;
      p:=act*el/20;
      f[2,mn]^:=f[2,mn]^-p*(st-so)*(10-5*c+2*d);
      f[3,mn]^:=f[3,mn]^-p*el*(st-so)*(10*(2*st+so)-10*c+3*d)/3;
      f[5,mn]^:=f[5,mn]^-p*(st-so)*(5*c-2*d);
      f[6,mn]^:=f[6,mn]^+p*el*(st-so)*(5*c-3*d)/3;
    end;{5}
  6: begin                          (**  ELEMENT IMPERFECTION  **)
      el:=eleng[mn]^;
      a:=emod[mn]^*zi[mn]^/(el*el*el);
      b:=area[mn]^*el*el/zi[mn]^;
      f[1,mn]^:=f[1,mn]^+a*b*act;
      f[2,mn]^:=f[2,mn]^+12*a*dist1-6*el*a*dist2;
      f[3,mn]^:=f[3,mn]^+6*el*a*dist1-2*el*el*a*dist2;
      f[4,mn]^:=f[4,mn]^-a*b*act;
      f[5,mn]^:=f[5,mn]^-12*a*dist1+6*el*a*dist2;
      f[6,mn]^:=f[6,mn]^+6*el*a*dist1-4*el*el*a*dist2;
    end;{6}
```

```
          7: begin                                  (** TEMPERATURE CHANGE **)
               p:=area[mn]^*emod[mn]^*cte[mn]^*act;
               f[1,mn]^:=f[1,mn]^+p;
               f[4,mn]^:=f[4,mn]^-p;
             end;{7}
        end;{case}
      end;{for}
      ASSEMF(f,q,c1,c2,mcode,na,ne);
   end
   else begin
     writeln(outfile,'    NO MEMBER ACTIONS');
   end;{if}
end;  {of MACT}


{*****************************************************************************
 *                                  LOAD                                     *
 *****************************************************************************}
procedure LOAD(var f:Force; var q:Loaddisp; area,emod,cte,eleng,c1,c2:
               Elemprops; mcode:Member; jcode:Joint; var na:Actions;
               ne,neq:integer);
{
    Initialize to zero the joint load vector, q, the local element
    (member) force vector, f, and the number of actions vector, na;
    call JLOAD and MACT.
}
var i,k,l:integer;
begin
   for k:=1 to neq do
      q[k]^:=0.0;
   for i:= 1 to ne do begin
      na[i]:=0;
      for l:=1 to 6 do
         f[l,i]^:=0.0;
   end;{for}
   JLOAD(q,jcode);
   MACT(f,q,area,zi,emod,cte,eleng,c1,c2,mcode,na,ne);
end;  {of LOAD}


{*****************************************************************************
 *                                  DATA                                     *
 *****************************************************************************}
procedure DATA(var f:Force; var x:Gcoords; var q:Loaddisp;
               var area, zi, emod, cte, eleng, c1, c2:Elemprops;
               var mcode:Member; var jcode:Joint; var minc:Incidence;
               var na:Actions; ne, nj:integer; var neq, mbd:integer;
               lc:integer; var nh:integer; var h:Hdat; fname:S14; date:S8);

{      For the first load condition, lc=1, call STRUCT and LOAD,
```

```
      for subsequent load conditions, lc > 1, call LOAD.
}
var i:integer;
begin
   if lc=1 then begin
      STRUCT(x,area,zi,emod,cte,eleng,c1,c2,mcode,jcode,minc,ne,nj,neq,mbd,
            nh,h,fname,date);
{  Allocate real matricies to the heap:     }
      for i:=1 to neq do begin
         new(q[i]);
         for j:=1 to (mbd+1) do
         new(ss[i,j]);
      end;{for}
   end;{if}
   for i:=1 to 4 do writeln(outfile);
   writeln(outfile);
   writeln(outfile,'    LOAD CONDITION ',lc:2);
   writeln(outfile,'    ----------------');
   LOAD(f,q,area,emod,cte,eleng,c1,c2,mcode,jcode,na,ne,neq);
end;  {of DATA}


{*****************************************************************************
 *                                  ELEMS                                    *
 *****************************************************************************}
procedure ELEMS(area,zi,emod,eleng,c1,c2:Elemprops; var g:Gstiff; n:integer);
{
     For element n, compute the global stiffness coefficients, g[7],
     defined in eqs. 3.64.
}
var  c1n,c2n,el,alfa,beta:real;
begin
   c1n:=c1[n]^;
   c2n:=c2[n]^;
   el:=eleng[n]^;
   alfa:=emod[n]^*zi[n]^/(el*el*el);
   beta:=area[n]^*el*el/zi[n]^;
   g[1]^:=alfa*(beta*c1n*c1n+12.0*c2n*c2n);
   g[2]^:=alfa*c1n*c2n*(beta-12.0);
   g[3]^:=alfa*(beta*c2n*c2n+12.0*c1n*c1n);
   g[4]^:=-alfa*6.0*el*c2n;
   g[5]^:=alfa*6.0*el*c1n;
   g[6]^:=alfa*4.0*el*el;
   g[7]^:=alfa*2.0*el*el;
end;  {of ELEMS}
```

```
(*****************************************************************************
 *                                ASSEMS                                     *
 *****************************************************************************)
procedure ASSEMS(var ss:Matrix; g:Gstiff; mcode:Member; mbd,n:integer);
{
     Initialize index by eq. 7.4; assign stiffness coefficients, g[1],
     of element n to the system stiffness band matrix, ss, by index,
     mcode, and eq. 6.7.
}
var  index:Indmat;  i,ie,j,je,k,l:integer;
begin

{ Initialize index:     }
   index[1,1]:=1;    index[1,2]:=2;    index[1,3]:=4;    index[1,4]:=-1;
   index[1,5]:=-2;   index[1,6]:=4;    index[2,1]:=2;    index[2,2]:=3;
   index[2,3]:=5;    index[2,4]:=-2;   index[2,5]:=-3;   index[2,6]:=5;
   index[3,1]:=4;    index[3,2]:=5;    index[3,3]:=6;    index[3,4]:=-4;
   index[3,5]:=-5;   index[3,6]:=7;    index[4,1]:=-1;   index[4,2]:=-2;
   index[4,3]:=-4;   index[4,4]:=1;    index[4,5]:=2;    index[4,6]:=-4;
   index[5,1]:=-2;   index[5,2]:=-3;   index[5,3]:=-5;   index[5,4]:=2;
   index[5,5]:=3;    index[5,6]:=-5;   index[6,1]:=4;    index[6,2]:=5;
   index[6,3]:=7;    index[6,4]:=-4;   index[6,5]:=-5;   index[6,6]:=6;

   for je:=1 to 6 do begin
      j:=mcode[je,n];
      if j <> 0 then begin
        for ie:=je to 6 do begin
           i:=mcode[ie,n];
           if i <> 0 then begin
              k:=i-j+1;
              l:=index[ie,je];
              if l > 0
                 then  ss[j,k]^:=ss[j,k]^+g[l]^
                 else  ss[j,k]^:=ss[j,k]^-g[-l]^;
           end;{if}
        end;{for}
     end;{if}
   end;{for}
end;  {of ASSEMS}


(*****************************************************************************
 *                                STIFF                                      *
 *****************************************************************************)
procedure STIFF(var ss:Matrix; area,zi,emod,eleng,c1,c2:Elemprops;
               mcode:Member; ne,neq,mbd:integer);
{
     Initialize the system stiffness matrix, ss, to zero; for
     each element call ELEMS and ASSEMS.
}
```

```
var  j,l,m,n:integer;  g:Gstiff;
begin
   for j:=1 to 7 do new(g[j]);
   m:=mbd+1;
   for j:=1 to neq do begin
      for l:=1 to m do
        ss[j,l]^:=0.0;
   end;{for}
   for n:=1 to ne do begin
      ELEMS(area,zi,emod,eleng,c1,c2,g,n);
      ASSEMS(ss,g,mcode,mbd,n);
   end;{for}
   for j:=1 to 7 do  dispose(g[j]);
end;  {of STIFF}


{*****************************************************************************
 *                                  MODIFY                                   *
 *****************************************************************************}
procedure MODIFY(var ss:Matrix; var q:Loaddisp; jcode:Joint;
                 neq,mbd,lc:integer);
{
     Impose prescribed joint displacements (at the system level) by
     modifying ss and q via the Penalty Method.
}
var   jnum,jdir,npjd,j,i:integer;   pfac,pnum,displ:real;
begin
   readln(infile,npjd);
   for i:=1 to 3 do writeln(outfile);
   if npjd <> 0  then begin
      readln(infile,pfac);
      pnum:=0.0;
      for j:=1 to neq do begin
        if ss[j,1]^ > pnum then
           pnum:=ss[j,1]^;
      end;{for}
      if lc=1 then
        pnum:=pfac*pnum;
      writeln(outfile,'    PRESCRIBED JOINT DISPLACEMENTS');
      writeln(outfile,'    PFAC= ',pfac:7:1,'     PNUM= ',pnum:14:4);
      writeln(outfile,'    Joint   Direction   Displacement');
      for i:=1 to npjd do begin
        readln(infile,jnum,jdir,displ);
        writeln(outfile,'     ',jnum:3,'        ',jdir:1,'         ',
                  displ:8:5);
        j:=jcode[jdir,jnum];
        ss[j,1]^:=pnum;
        q[j]^:=displ*pnum;
      end;{for}
   end
```

```
   else
      writeln(outfile,'    NO PRESCRIBED JOINT DISPLACEMENTS');
end;  (of MODIFY)


{*****************************************************************************
 *                                  SOLVE                                    *
 *****************************************************************************}
procedure SOLVE(var ss:Matrix; var q:Loaddisp; neq,mbd,lc:integer);
{
     For the first load condition, lc=1, perform forward reduction
     of the stiffness matrix, forward reduction of the load-
     displacement matrix, q, then solve for the unknowns;
     for subsequent load conditions, lc > 1, perform forward reduction
     of the load-displacement matrix, q, then solve for the unknowns.
}
var   i,j,k,l,m,n:integer;   c:real;
begin
   if lc=1 then begin
{    Forward reduction of stiffness matrix (Gauss Elimination)      }
     for n:=1 to neq do begin
        for l:=2 to (mbd+1) do begin
           if ss[n,l]^ <> 0.0 then begin
              i:=n+l-1;
              c:=ss[n,l]^/ss[n,1]^;
              j:=0;
              for k:=l to (mbd+1) do begin
                 j:=j+1;
                 ss[i,j]^:=ss[i,j]^-c*ss[n,k]^;
              end;(for)
              ss[n,l]^:=c;
           end;(if)
        end;(for)
     end;(for)
   end;(if)
{  Forward reduction of constants (Gauss Elimination)               }
   for n:=1 to neq do begin
      for l:=2 to (mbd+1) do begin
         if ss[n,l]^ <> 0.0 then begin
            i:=n+l-1;
            q[i]^:=q[i]^-ss[n,l]^*q[n]^;
         end;(if)
      end;(for)
      q[n]^:=q[n]^/ss[n,1]^;
   end;(for)
{  Solve for unknowns by back-substitution                          }
   for m:=2 to neq do begin
      n:=neq+1-m;
      for l:=2 to (mbd+1) do begin
         if ss[n,l]^ <> 0.0 then begin
```

```
            k:=n+1-1;
            q[n]^:=q[n]^-ss[n,l]^*q[k]^;
          end;{if}
       end;{for}
    end;{for}
end;  {of SOLVE}


{*****************************************************************************
 *                                SYSTEM                                     *
 *****************************************************************************}
procedure SYSTEM(var ss:Matrix; var q:Loaddisp; area,zi,emod,eleng,
                 c1,c2:Elemprops; mcode:Member; ne,neq,mbd,lc:integer);
{
     For the first load condition, lc=1, call STIFF and SOLVE; for
     subsequent load conditions, lc > 1, call SOLVE.
     Note:  the input arguement, q, stores the joint loads; the
            output arguement, q, stores the joint displacement.
}
begin
   if lc=1
     then STIFF(ss,area,zi,emod,eleng,c1,c2,mcode,ne,neq,mbd);
   MODIFY(ss,q,jcode,neq,mbd,lc);
   SOLVE(ss,q,neq,mbd,lc);
end;  {of SYSTEM}


{*****************************************************************************
 *                                ELEMF                                      *
 *****************************************************************************}
procedure ELEMF(var f:Force; q:Loaddisp; area,zi,emod,eleng,c1,c2:Elemprops;
                mcode:Member; i:integer);
{
      Compute the local forces in element i, f[6,i]:  determine the
      global element displacements, d[6], from the joint displacement
      vector, q, via mcode; compute the local forces at the a-end
      of the element by eqs. 2.34, 2.36, & 3.59; use equilibrium to
      compute the local forces at the b-end of the element and
      eq. 3.95 to compute the actual element forces.
}
var    c1i,c2i,el,alfa,beta,d14,d25,d36,f1,f2,f3:real;
       k,l:integer;       d:Disp;
begin
   for k:=1 to 6 do new(d[k]);
   c1i:=c1[i]^;
   c2i:=c2[i]^;
   el:=eleng[i]^;
   alfa:=emod[i]^*zi[i]^/(el*el*el);
   beta:=area[i]^*el*el/zi[i]^;
{
```

```
   GLOBAL ELEMENT DISPLACEMENTS:
   Actual Displacements:
}
   for l:=1 to 6 do begin
      k:=mcode[l,i];
      if k=0
         then d[l]^:=0.0
         else d[l]^:=q[k]^;
   end;{for}
{
   Relative Displacements:
}
   d14:=d[1]^-d[4]^;
   d25:=d[2]^-d[5]^;
   d36:=d[3]^+d[6]^;
{
   Forces at a-end due to end displacements
}
   f1:=alfa*beta*(c1i*d14+c2i*d25);
   f2:=6.0*alfa*(2.0*c1i*d25-2.0*c2i*d14+el*d36);
   f3:=2.0*alfa*el*(3.0*c1i*d25-3.0*c2i*d14+el*(d[3]^+d36));
{
   Actual element-end forces
}
   f[1,i]^:=f[1,i]^+f1;
   f[2,i]^:=f[2,i]^+f2;
   f[3,i]^:=f[3,i]^+f3;
   f[4,i]^:=f[4,i]^-f1;
   f[5,i]^:=f[5,i]^-f2;
   f[6,i]^:=f[6,i]^+el*f2-f3;
   for k:=1 to 6 do  dispose(d[k]);
end;  {of ELEMF}


{*****************************************************************************
 *                                JOINTF                                     *
 *****************************************************************************}
procedure JOINTF(f:Force; var p:Jfmatrix; c1,c2:Elemprops; minc:Incidence;
                 i:integer);
{
     Transform the local forces of element i, f[6,i], to global
     forces and assign them to the joint force matrix, p, by
     eqs. 2.29, 2.34, 2.37, and minc.
}
var   j,k:integer;
begin
   j:=minc[1,i];
   k:=minc[2,i];
   p[1,j]^:=p[1,j]^+FG(c1[i]^,-c2[i]^,f[1,i]^,f[2,i]^);
   p[2,j]^:=p[2,j]^+FG(c2[i]^,c1[i]^,f[1,i]^,f[2,i]^);
```

```
   p[3,j]^:=p[3,j]^+f[3,i]^;
   p[1,k]^:=p[1,k]^+FG(c1[i]^,-c2[i]^,f[4,i]^,f[5,i]^);
   p[2,k]^:=p[2,k]^+FG(c2[i]^,c1[i]^,f[4,i]^,f[5,i]^);
   p[3,k]^:=p[3,k]^+f[6,i]^;
end;  (of JOINTF)


(*****************************************************************************
 *                                FORCES                                     *
 *****************************************************************************)
procedure FORCES(var f:Force; var p:Jfmatrix; q:Loaddisp; area,zi,emod,
                 eleng,c1,c2:Elemprops; mcode:Member; minc:Incidence;
                 ne:integer);
{
     For each element I, call ELEMF and JOINTF.
}
var i:integer;
begin
   for i:=1 to ne do begin
      ELEMF(f,q,area,zi,emod,eleng,c1,c2,mcode,i);
      JOINTF(f,p,c1,c2,minc,i);
   end;(for)
end;  (of FORCES)


(*****************************************************************************
 *                                DRAWF                                      *
 *****************************************************************************)
procedure DRAWF(x:Gcoords; minc:Incidence; h:Hdat; fname:S14; date:S8;
                ne,nj,nh,lc:integer; var ix,iy:Dcoords; var xrange,sfx,
                minx,xshift,yrange,sfy,maxy,yshift:real);
{
    Draws the structure using input data:  x, global coordinates, minc,
    member incidences, h, hinge data, ne, number of elements, nj, number
    of joints, and nh, number of hinges.
}
var
   i,j,k,l:integer;
   miny,maxx:real;
begin
   clrscr;  hires;  hirescolor(1);
   gotoxy(1,1);
   write(fname);
   gotoxy(3,2);
   write(date);
   gotoxy(1,5);
   write('Load Case ',lc:1);
   minx:=1E9;  miny:=1E9;  maxx:=-1E9;  maxy:=-1E9;
   for i:=1 to nj do begin
      if x[1,i]^ < minx then minx:=x[1,i]^;
```

```
      if x[1,i]^ > maxx then maxx:=x[1,i]^;
      if x[2,i]^ < miny then miny:=x[2,i]^;
      if x[2,i]^ > maxy then maxy:=x[2,i]^;
    end;{for}
    xrange:=maxx-minx;  yrange:=maxy-miny;
    if xrange >= yrange then begin
      sfx:=1.0;
      sfy:=yrange/xrange;
    end
    else begin
      sfy:=1.0;
      sfx:=xrange/yrange;
    end;{if}
    xshift:=(1-sfx)/2;
    yshift:=(1-sfy)/2;
    for j:=1 to nj do begin
      if xrange <> 0
        then ix[j]:=round(350*(sfx*(x[1,j]^-minx)/xrange+xshift)+154)
        else ix[j]:=330;
      if yrange <> 0
        then iy[j]:=round(160*(sfy*(maxy-x[2,j]^)/yrange+yshift)+20)
        else iy[j]:=99;
    end;{for}
    for j:=1 to nj do begin
      plot(ix[j],iy[j],1);
      draw(ix[j]+1,iy[j]+1,ix[j]+1,iy[j]-1,1);
      draw(ix[j]-1,iy[j]+1,ix[j]-1,iy[j]-1,1);
      draw(ix[j]+1,iy[j]-1,ix[j]-1,iy[j]-1,1);
      draw(ix[j]+1,iy[j]+1,ix[j]-1,iy[j]+1,1);
    end;{for}
    for i:=1 to ne do begin
      j:=minc[1,i];
      k:=minc[2,i];
      draw(ix[j],iy[j],ix[k],iy[k],1);
    end;{for}
    if nh > 0 then begin
      for i:=1 to nh do begin
        j:=h[1,i];
        k:=h[2,i];
        l:=minc[k,j];
        circle(ix[l],iy[l],4,1);
      end;{for}
    end;{if}
    repeat          until keypressed;
  end;  {of DRAWF}
```

```
{*****************************************************************************
 *                                  DRAWDF                                   *
 *****************************************************************************}
procedure DRAWDF(x:Gcoords; minc:Incidence; h:Hdat; fname:S14; date:S8;
                 ne,nj,nh,lc:integer; u:Jdisp; var ix,iy:Dcoords; c1,c2,
                 eleng:Elemprops; xrange,sfx,minx,xshift,yrange,sfy,maxy,
                 yshift:real);
{
    Draws the deflected structure using input data:  ix and iy, drawing
    coordinates, minc, member incidences, h, hinge data, ne, number of
    elements, nj, number of joints, nh, number of hinges, and u, joint
    displacements.
}
var
   i,j,k,l,lx,ly,px,py:integer;
   a,b,c,c1i,c2i,d1,d2,d3,d4,d5,d6,r1,r1,r2,xi,ud,umax,vd:real;

   function MAXR(a,b:real):real;    {Returns the absolute maximum of two reals}
      begin  if abs(a) >= abs(b)
                then MAXR:=abs(a)
                else MAXR:=abs(b)                              end;     {of MAXR}
   function UDISP(d1,d4,xi:real):real;          {Calculates the u displacement}
     begin  UDISP:=d1+(d4-d1)*xi                               end;    {of UDISP}
   function VDISP(d2,d3,d5,d6,r1,xi:real):real; {Calculates the v displacement}
     begin  VDISP:=(2*(d2-d5)+r1*(d3+d6))*xi*xi*xi
                     +(3*(d5-d2)-r1*(2*d3+d6))*xi*xi
                       +r1*d3*xi+d2                            end;    {of VDISP}
   procedure GTL(var a,b,c1,c2:real);             {Transforms global to local}
     var    c,d:real;
     begin  c:=a; d:=b; a:=c*c1+d*c2; b:=d*c1-c*c2              end;      {of GTL}
   procedure LTG(var a,b,c1,c2:real);             {Transforms local to global}
     var    c,d:real;
     begin  c:=a; d:=b; a:=c*c1-d*c2; b:=d*c1+c*c2              end;      {of LTG}

begin
   umax:=abs(u[1,1]);
   for i:=1 to nj do begin
     for j:=1 to 2 do begin
        if abs(u[j,i]) > umax  then umax:=abs(u[j,i]);
     end;{for}
   end;{for}
   for i:=1 to ne  do begin
      j:=minc[1,i];     k:=minc[2,i];
     d1:=u[1,j];       d4:=u[1,k];
     d2:=u[2,j];       d5:=u[2,k];
     d3:=u[3,j];       d6:=u[3,k];
     c1i:=c1[i]^;      c2i:=c2[i]^;
     r1:=eleng[i]^;
     GTL(d1,d2,c1i,c2i);
     GTL(d4,d5,c1i,c2i);
```

```
    a:=3*(2*(d2-d5)+r1*(d3+d6));
    b:=2*(3*(d5-d2)-r1*(2*d3+d6));
    c:=r1*d3;
    if (a <> 0.0) and ((b*b-4*a*c) >=0.0)  then begin
       r1:=(-b+sqrt(b*b-4*a*c))/(2*a);
       r2:=(-b-sqrt(b*b-4*a*c))/(2*a);
       if (0.0 <= r1) and (r1 <= 1.0)  then begin
          ud:=UDISP(d1,d4,r1);
          vd:=VDISP(d2,d3,d5,d6,r1,r1);
          LTG(ud,vd,c1i,c2i);
          umax:=MAXR(umax,ud);
          umax:=MAXR(umax,vd);
       end;{if}
       if (0.0 <= r2) and (r2 <= 1.0)  then begin
          ud:=UDISP(d1,d4,r2);
          vd:=VDISP(d2,d3,d5,d6,r1,r2);
          LTG(ud,vd,c1i,c2i);
          umax:=MAXR(umax,ud);
          umax:=MAXR(umax,vd);
       end;{if}
    end;{if}
end;{for}
for j:=1 to nj do begin
    if umax <> 0 then begin
       ix[j]:=round(ix[j]+41*u[1,j]/umax);
       iy[j]:=round(iy[j]-19*u[2,j]/umax);
    end;{if}
end;{for}
clrscr;  hires;  hirescolor(1);
for j:=1 to nj do begin
    plot(ix[j],iy[j],1);
    draw(ix[j]+1,iy[j]+1,ix[j]+1,iy[j]-1,1);
    draw(ix[j]-1,iy[j]+1,ix[j]-1,iy[j]-1,1);
    draw(ix[j]+1,iy[j]-1,ix[j]-1,iy[j]-1,1);
    draw(ix[j]+1,iy[j]+1,ix[j]-1,iy[j]+1,1);
end;{for}
for i:=1 to ne do begin
    j:=minc[1,i];     k:=minc[2,i];
    if umax=0.0  then
      draw(ix[j],iy[j],ix[k],iy[k],1)
    else begin
       d1:=u[1,j];        d4:=u[1,k];
       d2:=u[2,j];        d5:=u[2,k];
       d3:=u[3,j];        d6:=u[3,k];
       c1i:=c1[i]^;       c2i:=c2[i]^;
       r1:=eleng[i]^;
       GTL(d1,d2,c1i,c2i);
       GTL(d4,d5,c1i,c2i);
       for l:=0 to 250  do begin
          xi:=l/250;
```

```
            ud:=UDISP(d1,d4,xi);
            vd:=VDISP(d2,d3,d5,d6,r1,xi);
            if xrange <> 0.0
               then px:=round(350*(sfx*(x[1,j]^-minx+xi*eleng[i]^*c1[i]^)/
                         xrange+xshift)+154+(41/umax)*(ud*c1[i]^-vd*c2[i]^))
               else px:=330+round((41/umax)*(ud*c1[i]^-vd*c2[i]^));
            if yrange <> 0.0
               then py:=round(160*(sfy*(maxy-x[2,j]^-xi*eleng[i]^*c2[i]^)/
                         yrange+yshift)+20+(-19/umax)*(vd*c1[i]^+ud*c2[i]^))
               else py:=99+round((-19/umax)*(vd*c1[i]^+ud*c2[i]^));
            if l <> 0  then draw(px,py,lx,ly,1);
            lx:=px;
            ly:=py;
         end;{for}
      end;{if}
   end;{for}
   if nh > 0 then begin
      for i:=1 to nh do begin
         j:=h[1,i];
         k:=h[2,i];
         l:=minc[k,j];
         circle(ix[l],iy[l],4,1);
      end;{for}
   end;{if}
   gotoxy(1,1);
   write(fname);
   gotoxy(3,2);
   write(date);
   gotoxy(1,5);
   write('Load Case ',lc:1);
   gotoxy(1,7);
   write('Deflected');
   gotoxy(1,8);
   write('Structure');
   hirescolor(4);
   repeat          until keypressed;
   textmode;
end;  (of DRAWDF)


{*************************************************************************
 *                                OUTPUT                                 *
 *************************************************************************}
procedure  OUTPUT(f:Force; p:Jfmatrix; q:Loaddisp; jcode:Joint;
                  ne, nj, nh, lc:integer; x:Gcoords; h:Hdat; fname:S14;
                  date:S8; minc:Incidence; c1,c2,eleng:Elemprops);
{
     Use joint displacement vector, q, and jcode to print joint
     displacements (including joint constraints); print local
     element forces, f(6,ne); print joint forces, p(3,nj);
```

```
      call DRAWF and DRAWDF.
}
var  i,j,k,l:integer;     u:Jdisp;     ix,iy:Dcoords;
     t,t1,t2,xrange,sfx,minx,xshift,yrange,sfy,maxy,yshift:real;
begin
   for i:=1 to 3 do writeln(outfile);
   writeln(outfile,'    LOCAL ELEMENT FORCES');
   writeln(outfile,'                              a-end',
                   '                              b-end');
   writeln(outfile,'             ------------------------------',
                   '  ------------------------------');
   writeln(outfile,'    Element  Direct-1   Direct-2   Direct-3   Direct-4',
                   '   Direct-5   Direct-6');
   for i:=1 to ne do begin
      write(outfile,'       ',i:3);
      for l:=1 to 6 do begin
         t:=f[l,i]^;
         write(outfile,' ',t:10:3);
      end;{for}
      writeln(outfile);
   end;{for}
   for i:=1 to 3 do writeln(outfile);
   writeln(outfile,'    JOINT DISPLACEMENTS');
   writeln(outfile,'    Joint  Direction-1  Direction-2  Direction-3');
   for j:=1 to nj do begin
      for l:=1 to 3 do begin
         k:=jcode[l,j];
         if k=0 then
            u[l,j]:=0.0
         else
            u[l,j]:=q[k]^;
      end;{for}
      writeln(outfile,'    ',j:3,'    ',u[1,j]:10:5,'   ',u[2,j]:10:5,'   ',
              u[3,j]:10:5);
   end;{for}
   for i:=1 to 3 do writeln(outfile);
   writeln(outfile,'    JOINT FORCES');
   writeln(outfile,'    Joint  Direction-1  Direction-2  Direction-3');
   for j:=1 to nj do begin
      t:=p[1,j]^;  t1:=p[2,j]^;  t2:=p[3,j]^;
      writeln(outfile,'     ',j:3,'    ',t:10:3,'   ',t1:10:3,'   ',t2:10:3);
   end;{for}
   DRAWF(x,minc,h,fname,date,ne,nj,nh,lc,ix,iy,xrange,sfx,minx,xshift,yrange,
         sfy,maxy,yshift);
   DRAWDF(x,minc,h,fname,date,ne,nj,nh,lc,u,ix,iy,c1,c2,eleng,xrange,sfx,minx,
          xshift,yrange,sfy,maxy,yshift);
end;  {of OUTPUT}
```

```
{*************************************************************************
 *                                RESULT                                 *
 *************************************************************************}
procedure RESULT(f:Force; var p:Jfmatrix; q:Loaddisp; area, zi, emod,
                 eleng, c1, c2:Elemprops; mcode:Member; jcode:Joint;
                 minc:Incidence; ne,nj,nh,lc:integer; x:Gcoords; h:Hdat;
                 fname:S14; date:S8);
{
     Initializes the joint force matrix, p, to zero; call
     FORCES and OUTPUT.
}
var  j,l:integer;
begin
   for j:=1 to nj do begin
      for l:=1 to 3 do begin
        new(p[l,j]);
        p[l,j]^:=0.0;
      end;{for}
   end;{for}
   FORCES(f,p,q,area,zi,emod,eleng,c1,c2,mcode,minc,ne);
   OUTPUT(f,p,q,jcode,ne,nj,nh,lc,x,h,fname,date,minc,c1,c2,eleng);
end;  {of RESULT}


{*************************************************************************
 *                                MAIN                                   *
 *************************************************************************

    Initialize parameters mx, mxneq; read and echo ne, nj, nlc; if ne
    and nj are less than or equal to mx, call for each load condition
    DATA, SYSTEM, and RESULT; else print error message and stop.
}
begin
   clrscr;
   textcolor(4);
   writeln('        Program  FRAME  was translated and enhanced  by  ',
           'Kenneth  R.');
   writeln('    Nuttall  as  part  of his research for an MS in  CE  at  ',
           'Virginia');
   writeln('    Polytechnic Institute and State University.  It was ',
           'translated to');
   writeln('    Turbo  Pascal  from FORTRAN as it is developed  in  Holzer,',
           '  S.M.');
   writeln('    1985.  Computer Analysis of Structures.  Elsevier,  New York',
           ', and');
   writeln('    taught in CE 4001 and CE 4002.  This version  of FRAME also ',
           'incor-');
   writeln('    porates  the extensions covered by  CE 4980 during Spring  ',
           'Quarter');
   writeln('    1986.   These  extensions are:   hinges at  members'' ends ',
```

```
            '(Holzer,');
   writeln('   Section 4.4) and geometric  imperfections (Holzer,  ',
            'Section 4.7).');
   writeln('   Prescribed joint displacements are modeled as degrees of ',
            'freedom,');
   writeln('   i.e.,  they are not constrained, then they are declared ',
            'under the');
   writeln('   appropriate load case.');
   writeln('                                                        ',
           '4 November 1986');
   writeln;
   writeln;
   writeln;
   writeln('Hit any key to continue.');
   repeat  until keypressed;
   writeln;
   writeln;
   writeln;
{  Open Data Files:  }
   write('Input data file:  '); readln(fname);
   assign(infile,fname);        reset(infile);
   repeat
      write('Write FRAME.OUT to the default drive (Y/N)?  ');
      readln(ch);
      if (upcase(ch) <> 'Y') and (upcase(ch) <> 'N')  then write(^G)
   until (upcase(ch)='Y') or (upcase(ch)='N');
   if upcase(ch)='Y'  then
      FRO:='FRAME.OUT'
   else begin
      repeat
         writeln;
         write('Which drive do you want to write FRAME.OUT to:  A, B, or ',
               'C?  ');
         readln(dr);
         if (upcase(dr)<>'A') and (upcase(dr)<>'B') and (upcase(dr)<>'C')
            then write(^G)
      until (upcase(dr)='A') or (upcase(dr)='B') or (upcase(dr)='C');
      FRO:=dr+':FRAME.OUT'
   end;
   assign(outfile,FRO);  rewrite(outfile);
   readln(infile,date);
   readln(infile,ne,nj,nlc);
   writeln(outfile,
   '*****************************************************************************');
   writeln(outfile,
   '*                       ',title:20       ,'                          *');
   writeln(outfile,
   '*                         (Nuttall, 1986)                           *');
   writeln(outfile,
   '*****************************************************************************');
```

```
   writeln(outfile);
   writeln(outfile,
   '  DATA FILE:  ',fname:12,'                              DATE:  ',date:8);
   writeln(outfile);writeln(outfile);
   writeln(outfile,'    ',units:36);
   for i:=1 to 3 do writeln(outfile);
   writeln(outfile,'    CONTROL VARIABLES');
   writeln(outfile,'    Number of elements       ',ne:4);
   writeln(outfile,'    Number of joints         ',nj:4);
   writeln(outfile,'    Number of load conditions',nlc:4);
   if (ne <= mx) and (nj <= mx) then begin
{   Allocate real matricies to the heap:     }
     for i:=1 to 6 do begin
        for j:=1 to ne do
          new(f[i,j]);
     end;{for}
     for i:=1 to 3 do begin
        for j:=1 to nj do
          new(x[i,j]);
     end;{for}
     for i:=1 to ne do begin
        new(area[i]);
        new(zi[i]);
        new(emod[i]);
        new(cte[i]);
        new(eleng[i]);
        new(c1[i]);
        new(c2[i]);
     end;{for}
     for lc:=1 to nlc do begin
        DATA(f,x,q,area,zi,emod,cte,eleng,c1,c2,mcode,jcode,minc,na,ne,nj,
             neq,mbd,lc,nh,h,fname,date);
        SYSTEM(ss,q,area,zi,emod,eleng,c1,c2,mcode,ne,neq,mbd,lc);
        RESULT(f,p,q,area,zi,emod,eleng,c1,c2,mcode,jcode,minc,ne,nj,nh,lc,
               x,h,fname,date)
     end;{for}
   end
   else begin
     for j:=1 to 3 do writeln(outfile);
     writeln(outfile,'ERROR MESSAGE:  at least ne or nj exceeds mx;');
     writeln(outfile,'                increase value of mx and change');
     writeln(outfile,'                mxneq to 3*(mx-1).');
   end;{if}
   close(outfile);
   close(infile);
end.  (of Program Frame)
```

Appendix 3

Computer Listing
for
Program PREP

```
program PREP;  (11-4-86)
type
   S2=string[2];     S3=string[3];     S4=string[4];     S6=string[6];
   S8=string[8];     S9=string[9];     S10=string[10];   S12=string[12];
   S14=string[14];   S18=string[18];   S24=string[24];   S25=string[25];
   S36=string[36];   S46=string[46];
   MIPtr=^MI;                                  (Member Incidences)
   MI=record
        mi,mi2:integer;
        nextmi:MIPtr
      end;
   TIPtr=^TI;                          (Joint Constraints & Hinges)
   TI=record
        TI1,TI2:integer;
        nextI:TIPtr
      end;
   XYPtr=^XY;                                  (Joint Coordinates)
   XY=record
        x1,x2:real;
        nextx:XYPtr
       end;
   T2Ptr=^T2;                             (All section properties)
   T2=record
        T21:real;
        next2:T2Ptr
      end;
   T3Ptr=^T3;      (Joint Loads & Prescribed Joint Displacements)
   T3=record
        T31,T32:integer;
        T33:real;
        next3:T3Ptr
      end;
   MLPtr=^ML;                                    (Member Actions)
   ML=record
        mln,mlt:integer;
        mld,mld1,mld2:real;
        nextml:MLPtr
      end;
   ControlPtr=^Control;             (Counters for each load case)
   Control=record
             connjl,conmat,connpjd:integer;
             conpfac:real;
             nextcon:ControlPtr
           end;

var
   fname                                :S14;
   Fmi,Lmi                              :MIPtr;
   Fjc,Ljc,Fh,Lh                        :TIPtr;
   Fxy,Lxy                              :XYPtr;
```

```
   Fa,La,Fzi,Lzi,Fe,Le,Fcte,Lcte        :T2Ptr;
   Fjl,Ljl,Fpjd,Lpjd                    :T3Ptr;
   Fmact,Lmact                          :MLPtr;
   Flccv,Llccv                          :ControlPtr;
   scv,smi,sjc,sh,sxy,sep,sid           :S3;
   fn                                   :S10;
   date                                 :S8;
   ch                                   :char;
   ne,nj,nlc,njc,nh,tnjl,tnmact,tnpjd,
   nminc,nxy,na,nzi,nemod,ncte,nlccv    :integer;
   ifn,ofn                              :text;
   frame                                :file;
   EH                                   :boolean;

{$I GRAPH.P }


{*****************************************************************************
 *                              BLANKxx(i,j)                                 *
 *****************************************************************************}
{ BLANKxx(i,j) writes xx blank spaces starting at column i on line j. }
procedure BLANK80(i:integer);
   begin   gotoxy(i,i);   clreol                              end;
procedure BLANK40(i,j:integer);
   begin   gotoxy(i,j);
   write('                                        ');         end;
procedure BLANK24(i,j:integer);
   begin   gotoxy(i,j);   write('                        ')  end;
procedure BLANK20(i,j:integer);
   begin   gotoxy(i,j);   write('                    ')      end;
procedure BLANK10(i,j:integer);
   begin   gotoxy(i,j);   write('          ')                end;
procedure BLANK5(i,j:integer);
   begin   gotoxy(i,j);   write('     ')                     end;


{*****************************************************************************
 *                              GTWx(i,j,k)                                  *
 *****************************************************************************}
{ GTWx(i,j,k) goes to screen coordinates (i,j) and writes k. }
procedure GTWI(i,j,k:integer);
   begin   gotoxy(i,j);   write(k:1)   end;
procedure GTW1(i,j:integer; c:char);
   begin   gotoxy(i,j);   write(c:1)   end;
procedure GTW2(i,j:integer; A:S2);
   begin   gotoxy(i,j);   write(A:2)   end;
procedure GTW3(i,j:integer; A:S3);
   begin   gotoxy(i,j);   write(A:3)   end;
procedure GTW4(i,j:integer; A:S4);
   begin   gotoxy(i,j);   write(A:4)   end;
```

```
procedure GTW6(i,j:integer; A:S6);
   begin   gotoxy(i,j);   write(A:6)   end;
procedure GTW8(i,j:integer; A:S8);
   begin   gotoxy(i,j);   write(A:8)   end;
procedure GTW9(i,j:integer; A:S9);
   begin   gotoxy(i,j);   write(A:9)   end;
procedure GTW10(i,j:integer; A:S10);
   begin   gotoxy(i,j);   write(A:10)   end;
procedure GTW12(i,j:integer; A:S12);
   begin   gotoxy(i,j);   write(A:12)   end;
procedure GTW18(i,j:integer; A:S18);
   begin   gotoxy(i,j);   write(A:18)   end;
procedure GTW24(i,j:integer; A:S24);
   begin   gotoxy(i,j);   write(A:24)   end;
procedure GTW25(i,j:integer; A:S25);
   begin   gotoxy(i,j);   write(A:25)   end;
procedure GTW36(i,j:integer; A:S36);
   begin   gotoxy(i,j);   write(A:36)   end;
procedure GTW46(i,j:integer; A:S46);
   begin   gotoxy(i,j);   write(A:46)   end;


{*****************************************************************************
 *                                 RTMM                                      *
 *****************************************************************************}
{ RTMM returns control to Main Menu. }
procedure RTMM;
begin   clrscr;         GTW24(29,9,'Returning to MAIN MENU. ');
        gotoxy(1,25);   delay(1500)                              end;


{*****************************************************************************
 *                                 BEL                                       *
 *****************************************************************************}
{ BEL writes ^G }
procedure BEL;     begin   write(^G)   end;


{*****************************************************************************
 *                                 BM                                        *
 *****************************************************************************}
{ BM writes bottom menus }
procedure BM;
begin
   GTW9(14,24,'F1-Change');       GTW6(36,24,'F2-Add');
   GTW9(58,24,'F3-Insert');       GTW9(14,25,'F4-Delete');
   GTW10(36,25,'F10-Return');     gotoxy(1,25);
end;
procedure BMFK(i:integer);
begin
```

```
   textcolor(31);        BEL;                GTW2(14,24,'F1');
   GTW2(36,24,'F2');     GTW2(58,24,'F3');   GTW2(14,25,'F4');
   GTW3(36,25,'F10');    gotoxy(1,25);       textcolor(i)
end;
procedure BMI;
begin   BLANK80(24);   BLANK80(25);   GTW6(61,24,'Insert');   end;
procedure BMC;
begin   BLANK80(24);   BLANK80(25);   GTW6(17,24,'Change');   end;
procedure BMD;
begin   BLANK80(24);   BLANK80(25);   GTW6(17,25,'Delete');   end;
procedure BMA;
begin   BLANK80(24);   BLANK80(25);   GTW3(39,24,'Add');   end;


{*****************************************************************************
 *                                READFILE                                   *
 *****************************************************************************}
procedure READFILE(var ne,nj,nlc:integer; var Fmi,Lmi:MIPtr; var Fjc,Ljc,
                   Fh,Lh:T1Ptr; var Fxy,Lxy:XYPtr; var Fa,La,Fzi,Lzi,Fe,Le,
                   Fcte,Lcte:T2Ptr; var Fjl,Ljl,Fpjd,Lpjd:T3Ptr;
                   var Fmact,Lmact:MLPtr; var Flccv,Llccv:ControlPtr;
                   var njc,nh,tnjl,tnmact,tnpjd,nminc,nxy,na,nzi,nemod,ncte,
                   nlccv:integer);
var  i,j,njl,nmat,npjd:integer;   pfac:real;
{
   Reads the file which is opened in OLDFILE and places it in the memory.
}
begin
   readln(ifn,ne,nj,nlc);                                  {Control Variables}
   for i:=1 to ne do begin                                 {Member Incidences}
      if i=1  then begin
         new(Fmi);
         Lmi:=Fmi
      end
      else begin
         new(Lmi^.nextmi);
         Lmi:=Lmi^.nextmi
      end;
      with Lmi^  do begin
         readln(ifn,m1,m2);
         nextmi:=nil
      end;
      minc:=minc+1
   end;
   readln(ifn,njc);                                        {Joint Constraints}
   for i:=1 to njc  do begin
      if i=1  then begin
         new(Fjc);
         Ljc:=Fjc
      end
```

```
   else begin
      new(Ljc^.next1);
      Ljc:=Ljc^.next1
   end;
   with Ljc^  do begin
      readln(ifn,T11,T12);
      next1:=nil
   end
end;
readln(ifn,nh);                                          {Hinges}
for i:=1 to nh  do begin
   if i=1  then begin
      new(Fh);
      Lh:=Fh
   end
   else begin
      new(Lh^.next1);
      Lh:=Lh^.next1
   end;
   with Lh^  do begin
      readln(ifn,T11,T12);
      next1:=nil
   end
end;
for i:=1 to nj  do begin                          {Joint Coordinates}
   if i=1  then begin
      new(Fxy);
      Lxy:=Fxy
   end
   else begin
      new(Lxy^.nextx);
      Lxy:=Lxy^.nextx
   end;
   with Lxy^  do begin
      readln(ifn,x1,x2);
      nextx:=nil
   end;
   nxy:=nxy+1
end;
for i:=1 to ne  do begin                          {Member Properties}
   if i=1  then begin
      new(Fa);
      La:=Fa;
      new(Fzi);
      Lzi:=Fzi;
      new(Fe);
      Le:=Fe;
      new(Fcte);
      Lcte:=Fcte
   end
```

```
    else begin
       new(La^.next2);
       La:=La^.next2;
       new(Lzi^.next2);
       Lzi:=Lzi^.next2;
       new(Le^.next2);
       Le:=Le^.next2;
       new(Lcte^.next2);
       Lcte:=Lcte^.next2
    end;
    with La^  do begin
       read(ifn,T21);
       next2:=nil
    end;
    na:=na+1;
    with Lzi^  do begin
       read(ifn,T21);
       next2:=nil
    end;
    nzi:=nzi+1;
    with Le^  do begin
       read(ifn,T21);
       next2:=nil
    end;
    nemod:=nemod+1;
    with Lcte^  do begin
       readln(ifn,T21);
       next2:=nil
    end;
    ncte:=ncte+1
 end;
 for i:=1 to nlc  do begin                          {Load Cases}
    pfac:=0.0;
    readln(ifn,njl);                                {Joint Loads}
    for j:=1 to njl  do begin
       if Fjl=nil  then begin
          new(Fjl);
          Ljl:=Fjl
       end
       else begin
          new(Ljl^.next3);
          Ljl:=Ljl^.next3
       end;
       with Ljl^  do begin
          readln(ifn,T31,T32,T33);
          next3:=nil
       end;
       tnjl:=tnjl+1
    end;
    readln(ifn,nmat);                              {Member Actions}
```

```
for j:=1 to nmat  do begin
   if Fmact=nil  then begin
      new(Fmact);
      Lmact:=Fmact
   end
   else begin
      new(Lmact^.nextml);
      Lmact:=Lmact^.nextml
   end;
   with Lmact^  do begin
      readln(ifn,mln,mlt,mld,mld1,mld2);
      nextml:=nil
   end;
   tnmact:=tnmact+1
end;
readln(ifn,npjd);                        {Prescribed Joint Displacements}
if npjd <> 0  then begin
   readln(ifn,pfac);
   for j:=1 to npjd  do begin
      if Fpjd=nil  then begin
         new(Fpjd);
         Lpjd:=Fpjd
      end
      else begin
         new(Lpjd^.next3);
         Lpjd:=Lpjd^.next3
      end;
      with Lpjd^  do begin
         readln(ifn,T31,T32,T33);
         next3:=nil
      end;
      tnpjd:=tnpjd+1
   end
end;
if i=1  then begin
   new(Flccv);
   Llccv:=Flccv
end
else begin
   new(Llccv^.nextcon);
   Llccv:=Llccv^.nextcon
end;
with Llccv^  do begin
   connjl:=njl;
   connmat:=nmat;
   connpjd:=npjd;
   compfac:=pfac;
   nextcon:=nil
end;
nlccv:=nlccv+1
```

```
   end;
   close(ifn);
end;


{*****************************************************************************
 *                                 OLDFILE                                   *
 *****************************************************************************}
procedure OLDFILE(var fn:S10; var date:S8; var ne,nj,nlc,njc,nh,tnjl,tnmact,
                  tnpjd,nminc,nxy,na,nzi,nemod,ncte,nlccv:integer; var Fmi,Lmi:
                  MIPtr; var Fjc,Ljc,Fh,Lh:T1Ptr; var Fxy,Lxy:XYPtr;
                  var Fa,La,Fzi,Lzi,Fe,Le,Fcte,Lcte:T2Ptr; var Fjl,Ljl,
                  Fpjd,Lpjd:T3Ptr; var Fmact,Lmact:MLPtr; var Flccv,Llccv:
                  ControlPtr);
var  i:integer;  od:S8;  inf:S14;  ok:boolean;
{
    Reads the name of the data file and the date; checks for an existing
    file of the same name; if file is found calls READFILE.
}
begin
   clrscr;
   textcolor(4);
   gotoxy(18,8);
   write('Enter File Name (1 to 8 characters):  ');
   readln(fn);
   gotoxy(33,10);
   write('Date:  ');
   readln(date);
   for i:=1 to 7 do writeln;
   inf:=fn+'.DAT';
   assign(ifn,inf);
   {$I-}  reset(ifn)  {$I+};
   ok:=(IOresult=0);
   if ok  then begin
      readln(ifn,od);
      gotoxy(30,14);
      write('Editing existing file.');
      gotoxy(21,15);
      write(inf:12,' was edited last on ',od:8);
      gotoxy(1,25);
      READFILE(ne,nj,nlc,Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,Fe,Le,
               Fcte,Lcte,Fjl,Ljl,Fpjd,Lpjd,Fmact,Lmact,Flccv,Llccv,njc,nh,
               tnjl,tnmact,tnpjd,nminc,nxy,na,nzi,nemod,ncte,nlccv);
   end
   else begin
      gotoxy(31,14);
      writeln('Creating a new file.');
      gotoxy(1,25)
   end;
   delay(2000)
```

```
end;


{*****************************************************************************
 *                                  CONTV                                    *
 *****************************************************************************}
procedure CONTV(fn:S10; date:S8; var ne,nj,nlc:integer);
var  fk:char;  i:integer;  EH,OK:boolean;
{
    Reads and edits the Control Variables, ne, number of elements,
    nj, number of joints, nlc, the number of load conditions.
}
   procedure CVFK;  (Flashes the function key indicators)
   begin
      gotoxy(26,15);   textcolor(31);   write(^G,'F1, F2, F3, or F10');
      gotoxy(1,25);    textcolor(12)
   end;  (flash F keys)

begin
   clrscr;          textcolor(12);
   gotoxy(9,1);     write(fn,'.DAT');
   gotoxy(32,1);    write('CONTROL VARIABLES');
   gotoxy(64,1);    write(date);
   gotoxy(31,5);    write('NE=',ne:2,'  NJ=',nj:2,'  NLC=',nlc:2);
   gotoxy(24,8);    write('Key   To Change');
   gotoxy(24,9);    write('---   -------------------------');
   gotoxy(24,10);   write('F1    The number of elements');
   gotoxy(24,11);   write('F2    The number of joints');
   gotoxy(24,12);   write('F3    The number of loading cases');
   gotoxy(24,13);   write('F10   Return to the MAIN MENU');
   repeat
     EH:=False;
     gotoxy(11,15);  write('Please use the F1, F2, F3, or F10 key to enter ',
                           'your choice.');
     gotoxy(1,25);
     repeat
        EH:=False;
        read(kbd,fk);
        EH:=(fk=#27);
        if EH and keypressed  then begin
           read(kbd,fk);
           case fk of
              #59:begin                                  (Number of Elements)
                    repeat
                      GTW3(38,19,'NE=');   GTW9(36,20,'(1 to 30)');
                      gotoxy(41,19);
                      {$I-}  readln(ne);  {$I+}
                      OK:=(IOresult=0);
                      if (ne < 1) or (ne > 30) or (not OK)  then begin
                         BEL;
```

```
            GTW46(17,22,
             'Parameter entered is out of range.  Try again.');
            BLANK10(41,19)
          end
          else begin
            gotoxy(34,5);  write(ne:2);
            for i:=19 to 22  do BLANK80(i)
          end
        until (ne >= 1) and (ne <= 30) and OK
      end;
#60:begin                                    (Number of Joints)
        repeat
          GTW3(38,19,'NJ=');   GTW9(36,20,'(1 to 30)');
          gotoxy(41,19);
          {$I-}  readln(nj);  {$I+}
          OK:=(IOresult=0);
          if (nj < 1) or (nj > 30) or (not OK)  then begin
            BEL;
            GTW46(17,22,
             'Parameter entered is out of range.  Try again.');
            BLANK10(41,19)
          end
          else begin
            gotoxy(41,5);  write(nj:2);
            for i:=19 to 22  do BLANK80(i)
          end
        until (nj >= 1) and (nj <= 30) and OK
      end;
#61:begin                                 (Number of Load Cases)
        repeat
          GTW10(38,19,'NLC=      ');
          gotoxy(42,19);
          {$I-}  readln(nlc);  {$I+}
          OK:=(IOresult=0);
          if not OK  then BEL
        until OK;
        gotoxy(49,5);         write(nlc:2);
        BLANK80(19)
      end;
#68:begin                                  (Return to MAIN MENU)
        if (ne <> 0) and (nj <> 0) and (nlc <> 0)  then
          RTMM
        else begin
          fk:=#67;
          gotoxy(22,19);
          write(^G,'All control variables must be defined.');
          delay(1500);
          BLANK80(19)
        end
      end;
```

```
            else CVFK                                    (All other input)
          end
        end
        else CVFK
     until ((fk=#59) or (fk=#60) or (fk=#61) or (fk=#68)) and EH
   until (fk=#68) and EH
end;


{*****************************************************************************
 *                                MENU                                       *
 *****************************************************************************}
procedure MENU(fn:S10; date:S8; scv,smi,sjc,sh,sxy,sep,sld:S3);
{
    Writes the main menu to the screen at the beginning of the program
    and after each editing subroutine returns to the main menu.
}
begin
   clrscr;
   textcolor(4);
   gotoxy(9,1);    write(fn:10,'.DAT');
   gotoxy(38,1);   write('MAIN MENU');
   gotoxy(64,1);   write(date:8);
   gotoxy(24,5);   write('Key');
   gotoxy(35,5);   write('Function');
   gotoxy(51,5);   write('Status');
   gotoxy(24,6);   write('---   ----------------   ------');
   gotoxy(24,7);   textcolor(12);  write('F1    Control Variables     ',scv:3);
   gotoxy(24,8);   textcolor(9);   write('F2    Member Incidence      ',smi:3);
   gotoxy(24,9);   textcolor(2);   write('F3    Joint Constraints     ',sjc:3);
   gotoxy(24,10);  textcolor(3);   write('F4    Hinges                ',sh:3);
   gotoxy(24,11);  textcolor(6);   write('F5    Joint Coordinates     ',sxy:3);
   gotoxy(24,12);  textcolor(5);   write('F6    Element Properties    ',sep:3);
   gotoxy(24,13);  textcolor(11);  write('F7    Load Data             ',sld:3);
   gotoxy(24,14);  textcolor(1);   write('F8    Display Frame');
   gotoxy(24,15);  textcolor(14);  write('F9    Exit to FRAME');
   gotoxy(24,16);  textcolor(15);  write('F10   Exit to DOS');
   gotoxy(15,19);  textcolor(4);   write('Please use the function keys to ',
                                         'enter your choice.');
   gotoxy(1,25)
end;


{*****************************************************************************
 *                              overlay ONE                                  *
 *****************************************************************************}
overlay procedure ONE(fn:S10; date:S8; var Fmi,Lmi:MIPtr; var Fjc,Ljc,Fh,Lh
                    :T1Ptr; var Fxy,Lxy:XYPtr; var Fa,La,Fzi,Lzi,Fe,Le,Fcte,
                    Lcte:T2Ptr; ne,nj,nlc:integer; var nmmc,njc,nh,nxy,na,
                    nzi,nemod,ncte:integer; fk:char);
```

```
{
   Allows for code larger then 64K.  Calls MEMINS, TYPE1, JTCOORDS,
   and ELPROP.
}
{$I MEMINS.PAS}
{$I TYPE1.PAS}
{$I JTCOORDS.PAS}
{$I ELPROP.PAS}
begin
  case fk of
     #60:MEMINS(fn,date,Fmi,Lmi,ne,nj,nminc);
     #61:TYPE1(fn,date,Fjc,Ljc,1,nj,njc);
     #62:TYPE1(fn,date,Fh,Lh,2,ne,nh);
     #63:JTCOORDS(fn,date,Fxy,Lxy,nj,nxy);
     #64:ELPROP(fn,date,Fa,La,Fzi,Lzi,Fe,Le,Fcte,Lcte,ne,na,nzi,nemod,ncte)
  end
end;


{*****************************************************************************
 *                              overlay TWO                                  *
 *****************************************************************************}
overlay procedure TWO(fn:S10; date:S8; var Fmi,Lmi:MIPtr; var Fjc,Ljc,Fh,Lh
                     :T1Ptr; var Fxy,Lxy:XYPtr; var Fa,La,Fzi,Lzi,Fe,Le,Fcte,
                     Lcte:T2Ptr; var Fjl,Ljl,Fpjd,Lpjd:T3Ptr; var Fmact,Lmact
                     :MLPtr; var Flccv,Llccv:ControlPtr; ne,nj,nlc,nminc,
                     njc,nh,nxy,na,nzi,nemod,ncte:integer; var tnjl,tnmact,
                     tnpjd,nlccv:integer; var fk:char);
{
   Allows for code larger than 64K.  Calls LOADS, DRAWS, RFRAME, and ETDOS.
}
{$I TYPE3.PAS}
{$I MACT.PAS}
{$I LOADS.PAS}
{$I DRAWS.PAS}
{$I ETDOS.PAS}
begin
  case fk of
     #65:LOADS(fn,date,Fjl,Ljl,Fpjd,Lpjd,Fmact,Lmact,Flccv,Llccv,ne,nj,nlc,
               tnjl,tnmact,tnpjd,nlccv);
     #66:DRAWS(fn,date,Fxy,Lxy,Fmi,Lmi,Fh,Lh,ne,nj,nh);
     #67:ETDOS(fn,date,Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,Fe,Le,Fcte,
              Lcte,Fjl,Ljl,Fpjd,Lpjd,Fmact,Lmact,Flccv,Llccv,1,ne,nj,nlc,
              njc,nh,nminc,nxy,na,nzi,nemod,ncte,nlccv,tnjl,tnmact,tnpjd,fk);
     #68:ETDOS(fn,date,Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,Fe,Le,Fcte,
              Lcte,Fjl,Ljl,Fpjd,Lpjd,Fmact,Lmact,Flccv,Llccv,2,ne,nj,nlc,
              njc,nh,nminc,nxy,na,nzi,nemod,ncte,nlccv,tnjl,tnmact,tnpjd,fk)
  end
end;
```

```
{*****************************************************************************
 *                                  PREP                                     *
 *****************************************************************************}
{
   Prepares the data file for program FRAME.COM.  Each category of data
   has at least one separate subroutine for entering data.  Old data files
   can be called, read into memory, edited, and saved for future runs
   from FRAME.COM.  FRAME.COM can be run from PREP.
}
  procedure PFK; {Flashes words function keys}
  begin
    gotoxy(30,19);      textcolor(31);      write(^G,'function keys');
    textcolor(4);       gotoxy(1,25);
  end;

begin
  Fmi:=nil;    Lmi:=nil;    Fjc:=nil;    Ljc:=nil;    Fh:=nil;     Lh:=nil;
  Fxy:=nil;    Lxy:=nil;    Fa:=nil;     La:=nil;     Fzi:=nil;    Lzi:=nil;
  Fe:=nil;     Le:=nil;     Fcte:=nil;   Lcte:=nil;   Fjl:=nil;    Ljl:=nil;
  Fpjd:=nil;   Lpjd:=nil;   Fmact:=nil;  Lmact:=nil;  Flccv:=nil;  Llccv:=nil;
  ne:=0;       nj:=0;       nlc:=0;      njc:=0;      nh:=0;       tnjl:=0;
  tnmact:=0;   tnpjd:=0;    nmine:=0;    nxy:=0;      na:=0;       nzi:=0;
  nemod:=0;    ncte:=0;     nlccv:=0;    scv:='No';   smi:='No';   sjc:='No';
  sh:='No';    sxy:='No';   sep:='No';   sld:='No';

  clrscr;
  textcolor(4);
  writeln;     writeln;     writeln;
  writeln('         Program PREP was developed by Kenneth R. Nuttall  as ',
          'part of');
  writeln('    his  research  for  an MS in CE at Virginia Polytechnic ',
          'Institute');
  writeln('    and  State  University.   It creates the data  file  required',
          '  by');
  writeln('    program  FRAME which was translated to Turbo Pascal from  ',
          'FORTRAN');
  writeln('    as  it is developed in  Holzer, S.M.  1985.  Computer ',
          'Analysis of');
  writeln('    Structures.  Elsevier,  New York,   and taught in CE 4001 ',
          'and  CE');
  writeln('    4002.   This  version of FRAME also incorporates  the  ',
          'extensions');
  writeln('    covered by CE 4980 during the Spring Quarter 1986.   These ',
          'exten-');
  writeln('    sions  are:   hinges at members'' ends (Holzer,  Section ',
          '4.4)  and');
  writeln('    geometric imperfections (Holzer,  Section 4.7).  ',
          'Prescribed joint');
```

```
writeln('    displacements are modeled as degrees of freedom,  i.e., ',
          'they are');
writeln('    not  constrained,  then  they are declared under the  ',
          'appropriate');
writeln('    load case.');
writeln('                                                  ',
          '4 November 1986');
GTW24(29,23,'Hit any key to continue.');
repeat   until keypressed;
OLDFILE(fn,date,ne,nj,nlc,njc,nh,tnjl,tnmact,tnpjd,nminc,nxy,na,nzi,nemod,
        ncte,nlccv,Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,Fe,Le,
        Fcte,Lcte,Fjl,Ljl,Fpjd,Lpjd,Fmact,Lmact,Flccv,Llccv);
fname:=fn+'.DAT';
if (ne=0) or (nj=0) or (nlc=0)  then begin
   CONTV(fn,date,ne,nj,nlc);
   scv:='Yes'
end;
repeat
   MENU(fn,date,scv,smi,sjc,sh,sxy,sep,sld);
   repeat
      EH:=False;
      read(kbd,ch);
      EH:=(ch=#27);
      if (ch=#27) and keypressed  then begin
        read(kbd,ch);
        case ch of
          #59:begin                                    {Control Variables}
                scv:='Yes';
                CONTV(fn,date,ne,nj,nlc)
              end;
          #60:begin                                    {Member Incidences}
                smi:='Yes';
                ONE(fn,date,Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,
                    Fe,Le,Fcte,Lcte,ne,nj,nlc,nminc,njc,nh,nxy,na,nzi,
                    nemod,ncte,ch);
              end;
          #61:begin                                    {Joint Constraints}
                sjc:='Yes';
                ONE(fn,date,Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,
                    Fe,Le,Fcte,Lcte,ne,nj,nlc,nminc,njc,nh,nxy,na,nzi,
                    nemod,ncte,ch);
              end;
          #62:begin                                          {Hinges}
                sh:='Yes';
                ONE(fn,date,Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,
                    Fe,Le,Fcte,Lcte,ne,nj,nlc,nminc,njc,nh,nxy,na,nzi,
                    nemod,ncte,ch);
              end;
          #63:begin                                    {Joint Coordinates}
                sxy:='Yes';
```

```
                ONE(fn,date,Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,
                    Fe,Le,Fcte,Lcte,ne,nj,nlc,nminc,njc,nh,nxy,na,nzi,
                    nemod,ncte,ch);
              end;
            #64:begin                                   {Member Properties}
                  sep:='Yes';
                  ONE(fn,date,Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,
                    Fe,Le,Fcte,Lcte,ne,nj,nlc,nminc,njc,nh,nxy,na,nzi,
                    nemod,ncte,ch);
              end;
            #65:begin                                     {Load Case Data}
                  sld:='Yes';
                  TWO(fn,date,Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,
                    Fe,Le,Fcte,Lcte,Fjl,Ljl,Fpjd,Lpjd,Fmact,Lmact,Flccv,
                    Llccv,ne,nj,nlc,nminc,njc,nh,nxy,na,nzi,nemod,ncte,
                    tnjl,tnmact,tnpjd,nlccv,ch);
              end;
            #66:TWO(fn,date,Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,
                  Fe,Le,Fcte,Lcte,Fjl,Ljl,Fpjd,Lpjd,Fmact,Lmact,Flccv,
                  Llccv,ne,nj,nlc,nminc,njc,nh,nxy,na,nzi,nemod,ncte,
                  tnjl,tnmact,tnpjd,nlccv,ch);              {Draws Frame}
            #67:TWO(fn,date,Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,
                  Fe,Le,Fcte,Lcte,Fjl,Ljl,Fpjd,Lpjd,Fmact,Lmact,Flccv,
                  Llccv,ne,nj,nlc,nminc,njc,nh,nxy,na,nzi,nemod,ncte,
                  tnjl,tnmact,tnpjd,nlccv,ch);
                       {Writes file, runs FRAME.COM, and returns to DOS}
            #68:TWO(fn,date,Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,
                  Fe,Le,Fcte,Lcte,Fjl,Ljl,Fpjd,Lpjd,Fmact,Lmact,Flccv,
                  Llccv,ne,nj,nlc,nminc,njc,nh,nxy,na,nzi,nemod,ncte,
                  tnjl,tnmact,tnpjd,nlccv,ch);
                                    {Writes file and returns to DOS}
            else PFK
          end
        end
        else PFK
      until ((ch=#59) or (ch=#60) or (ch=#61) or (ch=#62) or (ch=#63) or
            (ch=#64) or (ch=#65) or (ch=#66) or (ch=#67) or (ch=#68)) and
            EH
   until ((ch=#67) or (ch=#68)) and EH
end.
{*****************************************************************************
 *                              SUBSMI                                       *
 *****************************************************************************}
procedure SUBSMI(nminc:integer);
const  H1='Mem  a-end  b-end ';  H2='---  -----  ----- ';
{
   Writes the subheadings to the screen for member incidences.
}
begin
   BLANK80(4);    BLANK80(5);    BLANK80(24);    BLANK80(25);
```

```
   if nminc > 15  then begin
      GTW18(16,4,H1);  GTW18(48,4,H1);
      GTW18(16,5,H2);  GTW18(48,5,H2);
   end
   else begin
      GTW18(32,4,H1);  GTW18(32,5,H2);
   end
end;


(*****************************************************************************
 *                                LISTMI                                     *
 *****************************************************************************)
procedure LISTMI(nminc:integer; Fmi:MIPtr);
var  i,j:integer;  Tmi:MIPtr;
{
   Lists the member incidences in a tabular form.
}
begin
   if nminc=15  then begin
      for i:=6 to 20  do BLANK20(16,i);
      BLANK20(48,6)
   end;
   if nminc=16  then for i:=6 to 20  do BLANK20(32,i);
   Tmi:=Fmi;
   for i:=1 to nminc  do begin
      if nminc > 15  then begin
         if i <= 15
            then gotoxy(16,(5+i))
            else gotoxy(48,(i-10))
      end
      else gotoxy(32,(5+i));
      with Tmi^  do write(i:3,'    ',m1:2,'     ',m2:2);
      Tmi:=Tmi^.nextmi
   end;
   if Fmi=nil
      then BLANK20(32,6)
      else if nminc <= 15
         then BLANK20(32,(i+6))
         else BLANK20(48,(i-9))
end;


(*****************************************************************************
 *                                ADDMI                                      *
 *****************************************************************************)
procedure ADDMI(nj:integer; var Fmi,Lmi:MIPtr; var nminc:integer);
var  j:integer;  OK:boolean;
{
   Adds a record of the type MIPtr.
```

```
}
begin
   BMA;
   nminc:=nminc+1;
   gotoxy(26,22);  write('Member ',nminc:2,':  a-end=');
   if Fmi=nil  then begin
      new(Fmi);
      Lmi:=Fmi
   end
   else begin
      new(Lmi^.nextmi);
      Lmi:=Lmi^.nextmi
   end;
   with Lmi^  do begin
      repeat
         gotoxy(44,22);
         {$I-}  readln(m1);  {$I+}
         OK:=(IOresult=0);
         if (m1 < 1) or (m1 > nj) or (not OK)  then begin
            gotoxy(26,23);  write(^G,'Out of range.  A-end=1 to ',nj:2,'.');
            BLANK5(44,22)
         end
         else BLANK80(23)
      until (m1 >= 1) and (m1 <= nj) and OK;
      GTW6(48,22,'b-end=');
      repeat
         gotoxy(54,22);
         {$I-}  readln(m2);  {$I+}
         OK:=(IOresult=0);
         if (m2 < 1) or (m2 > nj) or (not OK)  then begin
            gotoxy(26,23);  write(^G,'Out of range.  B-end=1 to ',nj:2,'.');
            BLANK5(54,22)
         end
      until (m2 >= 1) and (m2 <= nj) and OK;
      nextmi:=nil
   end;
   for j:=22 to 23  do BLANK80(j);
end;


{*****************************************************************************
 *                                  DELMI                                    *
 *****************************************************************************}
procedure DELMI(var Fmi,Lmi:MIPtr; var nminc:integer);
var  j,k,kl:integer;  ans:char;  Tmi,Nlmi:MIPtr;  OK:boolean;
{
   Deletes a record of the type MIPtr.
}
begin
   BMD;
```

```
   Tmi:=Fmi;
   Nlmi:=Tmi;
   repeat
      GTW25(29,23,'Delete member number=    ');
      gotoxy(50,23);
      {$I-}  readln(k);  {$I+}
      OK:=(IOresult=0);
      if (k < 1) or (k > nminc) or (not OK)  then BEL
   until (k >= 1) and (k <= nminc) and OK;
   k1:=k-1;
   while k1 > 0  do begin
      k1:=k1-1;
      Nlmi:=Tmi;
      Tmi:=Tmi^.nextmi
   end;
   repeat
      for j:=22 to 23  do BLANK80(j);
      gotoxy(22,22);    write('Delete member ',k:1,'.  A-end=',Tmi^.m1:1,
                              ', B-end=',Tmi^.m2:1,'?');
      GTW8(36,23,'(Y/N):  ');
      readln(ans);
      if (upcase(ans) <> 'Y') and (upcase(ans) <> 'N')  then BEL;
      if upcase(ans)='Y'  then begin
         if Fmi=Lmi  then begin
            dispose(Fmi);
            Fmi:=nil;
            Lmi:=Fmi
         end
         else begin
            for j:=k to (nminc-1)  do begin
               if (k=1) and (j=1)
                  then Nlmi:=Tmi
                  else Nlmi:=Nlmi^.nextmi;
               Tmi:=Tmi^.nextmi;
               Nlmi^.m1:=Tmi^.m1;
               Nlmi^.m2:=Tmi^.m2
            end;
            Lmi:=Nlmi;
            dispose(Lmi^.nextmi);
            Lmi^.nextmi:=nil
         end;
         nminc:=nminc-1
      end
   until (upcase(ans)='Y') or (upcase(ans)='N');
   for j:=22 to 23  do BLANK80(j)
end;


{*********************:*************************************************************
 *                                CHANGEMI                                       *
```

```
{*************************************************************************}
procedure CHANGEMI(var Fmi:MIPtr; ne,nj:integer);
var  i,j:integer;  Tmi:MIPtr;  OK:boolean;
{
   Changes record of the type MIPtr.
}
begin
   BMC;
   Tmi:=Fmi;
   repeat
      gotoxy(30,22);   write('Change which member?       ');
      gotoxy(52,22);
      {$I-}  readln(i);  {$I+}
      OK:=(IOresult=0);
      if (i < 1) or (i > ne) or (not OK)  then BEL
   until (i >= 1) and (i <= ne) and OK;
   i:=i-1;
   while i > 0  do begin
      i:=i-1;
      Tmi:=Tmi^.nextmi
   end;
   with Tmi^  do begin
      repeat
         gotoxy(28,23);   write('Change a-end from ',m1:2,' to       ');
         gotoxy(52,23);
         {$I-}  readln(j);  {$I+}
         OK:=(IOresult=0);
         if (j < 1) or (j > nj) or (not OK)  then BEL
      until (1 <= j) and (j <= nj) and OK;
      m1:=j;
      repeat
         gotoxy(28,23);   write('Change b-end from ',m2:2,' to       ');
         gotoxy(52,23);
         {$I-}  readln(j);  {$I+}
         OK:=(IOresult=0);
         if (j < 1) or (j > nj) or (not OK)  then BEL
      until (1 <= j) and (j <= nj) and OK;
      m2:=j
   end;
   for i:=22 to 23  do BLANK80(i)
end;


{*************************************************************************
 *                                INMI                                   *
 *************************************************************************}
procedure INMI(var minc,nj:integer; var Fmi,Lmi:MIPtr);
var  Tmi:MIPtr;  i,i1,i2,j1,j2:integer;  OK:boolean;
{
   Inserts a record of the type MIPtr into the list of existing
```

```
    records.
}
begin
   BMI;
   new(Lmi^.nextmi);
   Lmi:=Lmi^.nextmi;
   Lmi^.nextmi:=nil;
   Tmi:=Fmi;
   repeat
      gotoxy(24,22);   write('Insert new member before member      ');
      gotoxy(57,22);
      {$I-}  readln(i);  {$I+}
      OK:=(IOresult=0);
      if (i < 1) or (i > nminc) or (not OK)  then BEL
   until (i >= 1) and (i <= nminc) and OK;
   i:=i-1;
   while i > 0  do begin
      i:=i-1;
      Tmi:=Tmi^.nextmi
   end;
   repeat
      GTW12(37,23,'A-end=      ');
      gotoxy(43,23);
      {$I-}  readln(i1);  {$I+}
      OK:=(IOresult=0);
      if (i1 < 1) or (i1 > nj) or (not OK)  then BEL
   until (1 <= i1) and (i1 <= nj) and OK;
   repeat
      GTW12(37,23,'B-end=      ');
      gotoxy(43,23);
      {$I-}  readln(i2);  {$I+}
      OK:=(IOresult=0);
      if (i2 < 1) or (i2 > nj) or (not OK)  then BEL
   until (1 <= i2) and (i2 <= nj) and OK;
   while Tmi^.nextmi <> nil  do begin
      with Tmi^  do begin
         j1:=m1;
         j2:=m2;
         m1:=i1;
         m2:=i2
      end;
      Tmi:=Tmi^.nextmi;
      i1:=j1;
      i2:=j2
   end;
   Tmi^.m1:=i1;
   Tmi^.m2:=i2;
   nminc:=nminc+1;
   for i:=22 to 23  do BLANK80(i);
end;
```

```
{*****************************************************************************
 *                                MEMINS                                     *
 *****************************************************************************}
procedure MEMINS(fn:S10; date:S8; var Fmi,Lmi:MIPtr; var ne,nj,nminc:integer);
var  i,j:integer;  EH:boolean;  fk,ans:char;
{
   Reads and edits the member incidences.
}
begin
   clrscr;        textcolor(9);
   gotoxy(9,1);   write(fn,'.DAT');
   gotoxy(32,1);  write('MEMBER INCIDENCES');
   gotoxy(64,1);  write(date);
   repeat
      EH:=False;
      SUBSMI(nminc);
      LISTMI(nminc,Fmi);
      if nminc=0  then for i:=1 to ne  do begin
         ADDMI(nj,Fmi,Lmi,nminc);
         SUBSMI(nminc);
         LISTMI(nminc,Fmi)
      end;
      if nminc < ne then begin
         j:=nminc+1;
         for i:=j to ne  do begin
            repeat
               GTW24(29,22,'Add or Insert?  (A/I):  ');
               readln(ans);
               BLANK80(22);
               if (upcase(ans) <> 'A') and (upcase(ans) <> 'I')  then BEL
            until (upcase(ans)='A') or (upcase(ans)='I');
            if upcase(ans)='A'
               then ADDMI(nj,Fmi,Lmi,nminc)
               else INMI(nminc,nj,Fmi,Lmi);
            SUBSMI(nminc);
            LISTMI(nminc,Fmi)
         end
      end
      else begin
         if nminc > ne  then begin
            j:=nminc-ne;
            gotoxy(27,22);  write('You must delete ',j:2,' members.');
            for i:=1 to j  do begin
               DELMI(Fmi,Lmi,nminc);
               SUBSMI(nminc);
               LISTMI(nminc,Fmi)
            end
         end
```

```
      end;
      BM;
      repeat
        read(kbd,fk);
        if (fk=#27) and keypressed  then begin
          EH:=True;
          read(kbd,fk);
          case fk of
            #59:CHANGEMI(Fmi,ne,nj);
            #60:ADDMI(nj,Fmi,Lmi,nminc);
            #61:INMI(nminc,nj,Fmi,Lmi);
            #62:DELMI(Fmi,Lmi,nminc);
            #68:RTNM;
            else BMFK(9)
          end
        end
        else BMFK(9)
      until ((fk=#59) or (fk=#60) or (fk=#61) or (fk=#62) or (fk=#68))
            and EH;
  until (fk=#68) and EH and (nminc=ne)
end;



{*****************************************************************************
 *                                  Flash                                    *
 *****************************************************************************}
{ Flash F keys }
procedure TIFK(j,i:integer);
begin
  BEL;  textcolor(31);   GTW2(1,8,'F1');   GTW2(1,9,'F2');
  if i=1  then GTW3(1,10,'F10')  else GTW2(1,10,'F3');
  textcolor((j+1));   gotoxy(1,25)
end;
procedure TIAFK;
begin
  textcolor(31);     GTW2(1,11,'F4');   GTW2(1,12,'F5');   GTW2(1,13,'F6');
  GTW2(1,14,'F7');   textcolor(2);      gotoxy(1,25)
end;


{*****************************************************************************
 *                                  CNNT1                                    *
 *****************************************************************************}
procedure CNNT1(Ft1:T1Ptr; var nnc:integer);
var   Tt1,Nlt1:T1Ptr;
{
   Counts the number of joints that have joint constraints or the number
   of members that have hinges.  Returns this value as nnjc.  Nnc may be
   less than or equal to njc or to nh.
```

```
}
begin
   nnc:=0;
   Tt1:=Ft1;
   if Tt1 <> nil  then begin
      repeat
         if Tt1^.next1 <> nil  then begin
            Nlt1:=Tt1;
            Tt1:=Tt1^.next1;
            if Nlt1^.T11 <> Tt1^.T11  then nnc:=nnc+1
         end
      until Tt1^.next1=nil;
      nnc:=nnc+1
   end
end;


{*****************************************************************************
 *                                SUBST1                                     *
 *****************************************************************************}
procedure SUBST1(j,nnc:integer);
var  H1,H2:S9;
{
    Writes the subheadings to the screen for joint constraints (j=1)
    or for hinges (j=2).
}
begin
   BLANK80(4);    BLANK80(5);    BLANK80(24);    BLANK80(25);
   if j=1  then begin
      H1:='JT  1 2 3';   H2:='--  - - -'
   end
   else begin
      H1:=' MEM  A B';   H2:=' ---  - -'
   end;
   if nnc > 15  then begin
      GTW9(28,4,H1);  GTW9(54,4,H1);
      GTW9(28,5,H2);  GTW9(54,5,H2)
   end
   else begin
      GTW9(36,4,H1);  GTW9(36,5,H2);
   end
end;


{*****************************************************************************
 *                                LIST1                                      *
 *****************************************************************************}
procedure LIST1(Ft1:T1Ptr; j,nnc:integer);
var   i,p,ix,iy:integer;   Tt1:T1Ptr;
{
```

```
   Lists the joint constraints (j=1) or the hinges (j=2) in tabular form.
}
begin
   if nnc=15  then begin
      for i:=6 to 20  do BLANK10(25,i);
      BLANK10(54,6)
   end;
   if nnc=16  then for i:=6 to 20  do BLANK10(37,i);
   p:=0;
   Tt1:=Ft1;
   for i:=1 to nnc  do begin
      if nnc > 15  then begin
         if i <= 15  then begin
            if j=1
               then ix:=28
               else ix:=29;
            iy:=i+5
         end
         else begin
            if j=1
               then ix:=54
               else ix:=55;
            iy:=i-10
         end
      end
      else begin
         if j=1
            then ix:=36
            else ix:=37;
         iy:=i+5
      end;
      gotoxy(ix,iy);   write(Tt1^.T11:2,'         ');
      repeat
         if j=1  then begin
            case Tt1^.T12 of
               1:GTW1((ix+4),iy,'1');
               2:GTW1((ix+6),iy,'2');
               3:GTW1((ix+8),iy,'3')
            end
         end
         else if Tt1^.T12=1
            then GTW1((ix+5),iy,'A')
            else GTW1((ix+7),iy,'B');
         p:=Tt1^.T11;
         Tt1:=Tt1^.next1
      until p <> Tt1^.T11
   end;
   if Ft1=nil
      then BLANK10(36,6)
      else BLANK10(ix,(iy+1))
```

```
end;


{*****************************************************************************
 *                                  PT1                                      *
 *****************************************************************************}
procedure PT1(var Ft1,Lt1:T1Ptr; j,no,i:integer; var nc:integer);
var   Tt1:T1Ptr;   i1,i2,j1,j2:integer;   c:char;
{
    Searches the linked list for the joint constraint which matches or is
    larger than the given joint constraint or the hinge which matches or is
    larger then the given hinge.  The new data is then inserted in the
    appropriate place.
}
begin
   if Ft1=nil  then begin
      new(Ft1);
      Lt1:=Ft1;
      with Lt1^  do begin
         T11:=no;
         T12:=i;
         next1:=nil
      end;
      nc:=nc+1;
   end
   else begin
      Tt1:=Ft1;
      while (Tt1^.T11 <= no) and (Tt1^.next1 <> nil) and
        ((Tt1^.T11 <> no) or (Tt1^.T12 < i))  do Tt1:=Tt1^.next1;
      if (Tt1^.T11 <> no) or (Tt1^.T12 <> i)  then begin
         new(Lt1^.next1);
         Lt1:=Lt1^.next1;
         Lt1^.next1:=nil;
         nc:=nc+1;
         if ((Tt1^.T11=no) and (Tt1^.T12 > i)) or (Tt1^.T11 > no)  then begin
            i1:=no;
            i2:=i;
            while Tt1^.next1 <> nil  do begin
               with Tt1^  do begin
                  j1:=T11;
                  j2:=T12;
                  T11:=i1;
                  T12:=i2
               end;
               Tt1:=Tt1^.next1;
               i1:=j1;
               i2:=j2
            end;
            Tt1^.T11:=i1;
            Tt1^.T12:=i2;
```

```
         end
         else begin
            Lt1^.T11:=no;
            Lt1^.T12:=i
         end
      end
      else begin
         gotoxy(17,24);
         if j=1  then
            write('Joint ',no:1,' is already constrained in direction ',
                  i:1,'.')
         else begin
            if i=1
               then c:='A'
               else c:='B';
            write('Member ',no:1,' already has a hinge at end ',c:1,'.')
         end;
         delay(1500)
      end
   end
end;


{*****************************************************************************
 *                                ADDT1                                      *
 *****************************************************************************}
procedure ADDT1(var Ft1,Lt1:T1Ptr; j,cv:integer; var nc:integer);
var   i,no:integer;   EH,OK:boolean;   fk:char;   w:S6;
{
    Reads no, the joint to be constrained (j=1) or the member that has
    a hinge on at least one end (j=2), and the type of joint constraint
    desired (j=1) or the end on which the hinge is to be placed.  Calls
    PT1 to add to the linked list.
}
begin
   for i:=8 to 10  do BLANK10(1,i);     GTW3(5,6,'Add');
   repeat
      EH:=False;
      gotoxy(24,23);
      if j=1
         then w:='joint '
         else w:='member';
      write('Enter ',w:6,' number (1 to ',cv:1,'):        ');
      gotoxy(56,23);
      {$I-}  readln(no);  {$I+}
      OK:=(IOresult=0);
      if (no < 1) or (no > cv) or (not OK)  then BEL
   until (no >= 1) and (no <= cv) and OK;
   BLANK80(23);
   if j=1  then begin
```

```
      GTW12(1,8,'F1- Fixed   ');            GTW12(1,9,'F2- Pinned   ');
      GTW18(1,10,'F3- 1-3 Constraint');     GTW18(1,11,'F4- 2-3 Constraint');
      GTW18(1,12,'F5- 1 Constraint  ');     GTW18(1,13,'F6- 2 Constraint  ');
      GTW18(1,14,'F7- 3 Constraint  ')
   end
   else begin
      GTW12(1,8,'F1-A end    ');      GTW12(1,9,'F2-B end    ');
      GTW12(1,10,'F3-Both ends')
   end;
   gotoxy(1,25);
   repeat
      read(kbd,fk);
      if (fk=#27) and keypressed  then begin
         EH:=True;
         read(kbd,fk);
         if j=1  then begin
            case fk of
               #59:for i:=1 to 3  do PT1(Ft1,Lt1,j,no,i,nc);
               #60:for i:=1 to 2  do PT1(Ft1,Lt1,j,no,i,nc);
               #61:begin
                     PT1(Ft1,Lt1,j,no,1,nc);
                     PT1(Ft1,Lt1,j,no,3,nc)
                   end;
               #62:for i:=2 to 3 do PT1(Ft1,Lt1,j,no,i,nc);
               #63:PT1(Ft1,Lt1,j,no,1,nc);
               #64:PT1(Ft1,Lt1,j,no,2,nc);
               #65:PT1(Ft1,Lt1,j,no,3,nc);
               else begin
                  T1FK(1,2);   T1AFK
               end
            end
         end
         else begin
            case fk of
               #59:PT1(Ft1,Lt1,j,no,1,nc);
               #60:PT1(Ft1,Lt1,j,no,2,nc);
               #61:for i:=1 to 2  do PT1(Ft1,Lt1,j,no,i,nc);
               else T1FK(j,2)
            end
         end
      end
      else begin
         T1FK(j,2);
         if j=1 then T1AFK
      end
   until EH and ((fk=#59) or (fk=#60) or (fk=#61))
            or ((j=1) and ((fk=#62) or (fk=#63) or (fk=#64) or (fk=#65)));
   BLANK5(4,6)
end;
```

```
{*****************************************************************************
 *                                  DELT1                                    *
 *****************************************************************************}
procedure DELT1(var Ft1,Lt1:T1Ptr; j,cv:integer; var nc:integer);
var   ic,l,k,k1:integer;   Tt1,Nlt1:T1Ptr;   w:S6;   c:char;   OK:boolean;
{
   Positions the linked list of joint constraints (j=1) or hinges (j=2)
   at the proper record to be deleted then it shuffles the data down the
   records and disposes of the last record.
}
begin
   for l:=8 to 10  do BLANK10(1,l);     GTW6(5,9,'Delete');
   ic:=0;
   Tt1:=Ft1;
   repeat
      gotoxy(24,23);
      if j=1
         then w:='joint '
         else w:='member';
      write('Enter ',w:6,' number (1 to ',cv:1,'):        ');
      gotoxy(56,23);
      {$I-}  readln(k);  {$I+}
      OK:=(IOresult=0);
      if (k < 1) or (k > cv) or (not OK)  then BEL
   until (k >= 1) and (k <= cv) and OK;
   BLANK80(23);
   if j=1  then begin
      repeat
         gotoxy(22,23);  write('Enter direction number (1 to 3):        ');
         gotoxy(56,23);
         {$I-}  readln(k1);  {$I+}
         OK:=(IOresult=0);
         if (k1 < 1) or (k1 > 3) or (not OK)  then BEL
      until (k1 >= 1) and (k1 <= 3) and OK;
   end
   else begin
      repeat
         gotoxy(26,23);   write('Enter member end (A or B):       ');
         gotoxy(54,23);   readln(c);
         if (upcase(c)='A') or (upcase(c)='B')  then begin
            if upcase(c)='A'
               then k1:=1
               else k1:=2
         end
         else BEL
      until (upcase(c)='A') or (upcase(c)='B')
   end;
   BLANK80(23);
   while (Tt1^.T11 <= k) and ((Tt1^.T11 <> k) or (Tt1^.T12 <> k1))
```

```
         and (Tt1^.next1 <> nil)  do begin
      ic:=ic+1;
      Nlt1:=Tt1;
      Tt1:=Tt1^.next1
    end;
    if (Tt1^.T11=k) and (Tt1^.T12=k1)  then begin
       if Lt1 <> Ft1  then begin
          ic:=ic+1;
          for l:=ic to (nc-1)  do begin
             if (ic=1) and (l=1)
                then Nlt1:=Tt1
                else Nlt1:=Nlt1^.next1;
             Tt1:=Tt1^.next1;
             Nlt1^.T11:=Tt1^.T11;
             Nlt1^.T12:=Tt1^.T12
          end;
          Lt1:=Nlt1;
          dispose(Lt1^.next1);
          Lt1^.next1:=nil
       end
       else begin
          dispose(Ft1);
          Ft1:=nil;
          Lt1:=Ft1
       end;
       nc:=nc-1
    end
    else begin
       gotoxy(20,24);
       if j=1
          then write('Joint ',k:1,' is not constrained in direction ',k1:1,'.')
          else write('Member ',k:1,' does not have a hinge at end ',c:1,'.');
       delay(1500)
    end
end;


{*****************************************************************************
 *                                  TYPE1                                    *
 *****************************************************************************}
procedure TYPE1(fn:S10; date:S8; var Ft1,Lt1:T1Ptr; j,cv:integer;
                var nc:integer);
var   Tt1:T1Ptr;   i,nnc:integer;   fk:char;   EH:boolean;
{
    Compiles and edits the joint constraint data (j=1) or the hinge
    data (j=2).  Calls CNNT1, SUBST1, LIST1, ADDT1, and DELT1.
}
begin
   clrscr;         textcolor((j+1));
   gotoxy(9,1);    write(fn,'.DAT');
```

```
gotoxy(64,1);   write(date);
if j=1  then begin
   gotoxy(32,1);   write('JOINT CONSTRAINTS')
end
else begin
   gotoxy(37,1);   write('HINGES')
end;
if Ft1 <> nil  then begin
   while Lt1^.T11 > cv  do begin
     case nc of
        1:begin
            dispose(Lt1);
            Ft1:=nil;
            Lt1:=Ft1
          end;
        2:begin
            Lt1:=Ft1;
            dispose(Lt1^.next1);
            Lt1^.next1:=nil
          end;
        else begin
            Tt1:=Ft1;
            for i:=1 to (nc-2)  do Tt1:=Tt1^.next1;
            Lt1:=Tt1;
            dispose(Lt1^.next1);
            Lt1^.next1:=nil
        end
     end;
     nc:=nc-1
   end
end;
repeat
   EH:=False;
   CNNT1(Ft1,nnc);
   SUBST1(j,nnc);
   LIST1(Ft1,j,nnc);
   if (nc=0) and (j=1)  then begin
      ADDT1(Ft1,Lt1,j,cv,nc);
      nnc:=1;
      LIST1(Ft1,j,nnc)
   end;
   GTW12(1,8,'F1- Add     ');           GTW12(1,9,'F2- Delete  ');
   GTW18(1,10,'F10-Return       ');
   for i:=11 to 14  do BLANK20(1,i);
   BLANK80(23);
   BLANK80(24);
   gotoxy(1,25);
   repeat
      read(kbd,fk);
      if (fk=#27) and keypressed  then begin
```

```
            EH:=True;
            read(kbd,fk);
            case fk of
               #59:ADDT1(Ft1,Lt1,j,cv,nc);
               #60:DELT1(Ft1,Lt1,j,cv,nc);
               #68:RTMM;
               else T1FK(j,1)
            end
         end
         else T1FK(j,1)
      until ((fk=#59) or (fk=#60) or (fk=#68)) and EH;
   until (fk=#68) and EH
end;



{*****************************************************************************
 *                                SUBSXY                                     *
 *****************************************************************************}
procedure SUBSXY(nxy:integer);
const  H1='JT     X1         X2     ';  H2='--  ---------  ---------';
{
   Writes the subheadings to the screen for joint coordinates.
}
begin
   BLANK80(4);    BLANK80(5);    BLANK80(24);    BLANK80(25);
   if nxy > 15  then begin
      GTW24(12,4,H1);  GTW24(46,4,H1);
      GTW24(12,5,H2);  GTW24(46,5,H2);
   end
   else begin
      GTW24(29,4,H1);  GTW24(29,5,H2);
   end
end;



{*****************************************************************************
 *                                LISTXY                                     *
 *****************************************************************************}
procedure LISTXY(nxy:integer; Fxy:XYPtr);
var  i,j:integer;  Txy:XYPtr;
{
   Lists the joint coordinates in a tabular form.
}
begin
   if nxy=15  then begin
      for i:=6 to 20  do BLANK24(12,i);
      BLANK24(46,6)
   end;
   if nxy=16  then for i:=6 to 20  do BLANK24(29,i);
```

```
   Txy:=Fxy;
   for i:=1 to nxy  do begin
      if nxy > 15  then begin
         if i <= 15
            then gotoxy(12,(5+i))
            else gotoxy(46,(i-10))
      end
      else gotoxy(29,(5+i));
      with Txy^  do write(i:2,'  ',x1:9:4,'  ',x2:9:4);
      Txy:=Txy^.nextx
   end;
   if Fxy=nil
      then BLANK24(29,6)
      else if nxy <= 15
         then BLANK24(29,i+6)
         else BLANK24(46,i-9);
end;


{*****************************************************************************
 *                                 ADDXY                                     *
 *****************************************************************************}
procedure ADDXY(var Fxy,Lxy:XYPtr; var nxy:integer);
var  j:integer;  OK:boolean;
{
    Adds a record of the type XYPtr.
}
begin
   BMA;
   nxy:=nxy+1;
   gotoxy(22,22);  write('Joint ',nxy:1,':  ');
   if Fxy=nil  then begin
      new(Fxy);
      Lxy:=Fxy
   end
   else begin
      new(Lxy^.nextx);
      Lxy:=Lxy^.nextx
   end;
   with Lxy^  do begin
      repeat
         GTW18(33,22,'X1=              ');
         gotoxy(36,22);
         {$I-}  readln(x1);  {$I+}
         OK:=(IOresult=0);
         if not OK  then BEL
      until OK;
      repeat
         GTW18(47,22,'X2=              ');
         gotoxy(50,22);
```

```
        {$I-} readln(x2); {$I+}
        OK:=(IOresult=0);
        if not OK then BEL
      until OK;
      nextx:=nil
    end;
    BLANK80(22);
end;


{*****************************************************************************
 *                                  DELXY                                    *
 *****************************************************************************}
procedure DELXY(var Fxy,Lxy:XYPtr; var nxy:integer);
var  j,k,k1:integer;  ans:char;  Txy,Nlxy:XYPtr;  OK:boolean;
{
   Deletes a record of the type XYPtr.
}
begin
   BMD;
   Txy:=Fxy;
   repeat
      GTW25(30,23,'Delete joint number=     ');
      gotoxy(50,23);
      {$I-} readln(k); {$I+}
      OK:=(IOresult=0);
      if (k < 1) or (k > nxy) or (not OK) then BEL
   until (k >= 1) and (k <= nxy) and OK;
   k1:=k-1;
   while k1 > 0 do begin
      k1:=k1-1;
      Nlxy:=Txy;
      Txy:=Txy^.nextx
   end;
   repeat
      for j:=22 to 23 do BLANK80(j);
      gotoxy(19,22);    write('Delete joint ',k:1,', X1=',Txy^.x1:9:4,
                              ', X2=',Txy^.x2:9:4,'?');
      GTW8(36,23,'(Y/N):  ');
      readln(ans);
      if (upcase(ans) <> 'Y') and (upcase(ans) <> 'N') then BEL;
      if upcase(ans)='Y' then begin
         for j:=k to (nxy-1) do begin
            if (k=1) and (j=1)
               then Nlxy:=Txy
               else Nlxy:=Nlxy^.nextx;
            Txy:=Txy^.nextx;
            Nlxy^.x1:=Txy^.x1;
            Nlxy^.x2:=Txy^.x2
         end;
```

```
        Lxy:=Nlxy;
        dispose(Lxy^.nextx);
        Lxy^.nextx:=nil;
        nxy:=nxy-1
      end
   until (upcase(ans)='Y') or (upcase(ans)='N');
   for j:=22 to 23  do BLANK80(j)
end;


{*****************************************************************************
 *                                CHANGEXY                                   *
 *****************************************************************************}
procedure CHANGEXY(var Fxy:XYPtr; nj:integer);
var  i:integer;  Txy:XYPtr;  x:real;  OK:boolean;
{
   Changes record of the type XYPtr.
}
begin
   BMC;
   Txy:=Fxy;
   repeat
      GTW25(29,22,'Change which joint?      ');
      gotoxy(50,22);
      {$I-}  readln(i);  {$I+}
      OK:=(IOresult=0);
      if (i < 1) or (i > nj) or (not OK)  then BEL
   until (i >= 1) and (i <= nj) and OK;
   i:=i-1;
   while i > 0  do begin
      i:=i-1;
      Txy:=Txy^.nextx
   end;
   with Txy^  do begin
      repeat
         gotoxy(22,23);
         write('Change X1 from ',x1:9:4,' to               ');
         gotoxy(50,23);
         {$I-}  readln(x);  {$I+}
         OK:=(IOresult=0);
         if not OK  then BEL
      until OK;
      x1:=x;
      repeat
         gotoxy(22,23);
         write('Change X2 from ',x2:9:4,' to               ');
         gotoxy(50,23);
         {$I-}  readln(x);  {$I+}
         OK:=(IOresult=0);
         if not OK  then BEL
```

```
      until OK;
      x2:=x
    end;
   for i:=22 to 23  do BLANK80(i)
end;


{*****************************************************************************
 *                                  INXY                                     *
 *****************************************************************************}
procedure INXY(var nxy:integer; var Fxy,Lxy:XYPtr);
var  Txy:XYPtr;  i:integer;  w1,w2,z1,z2:real;  OK:boolean;
{
    Inserts a record of the type XYPtr into the list of existing
    records.
}
begin
   BMI;
   new(Lxy^.nextx);
   Lxy:=Lxy^.nextx;
   Lxy^.nextx:=nil;
   Txy:=Fxy;
   repeat
      gotoxy(25,22);   write('Insert new joint before joint      ');
      gotoxy(55,22);
      {$I-}  readln(i);  {$I+}
      OK:=(IOresult=0);
      if (i < 1) or (i > nj) or (not OK)  then BEL
   until (i >= 1) and (i <= nxy) and OK;
   i:=i-1;
   while i > 0  do begin
      i:=i-1;
      Txy:=Txy^.nextx
   end;
   repeat
      GTW18(35,23,'X1=              ');
      gotoxy(38,23);
      {$I-}  readln(w1);  {$I+}
      OK:=(IOresult=0);
      if not OK  then BEL
   until OK;
   repeat
      GTW18(35,23,'X2=              ');
      gotoxy(38,23);
      {$I-}  readln(w2);  {$I+}
      OK:=(IOresult=0);
      if not OK  then BEL
   until OK;
   while Txy^.nextx <> nil  do begin
      with Txy^  do begin
```

```
         z1:=x1;
         z2:=x2;
         x1:=w1;
         x2:=w2
      end;
      Txy:=Txy^.nextx;
      w1:=z1;
      w2:=z2
   end;
   Txy^.x1:=w1;
   Txy^.x2:=w2;
   nxy:=nxy+1;
   for i:=22 to 23  do BLANK80(i);
end;


{*****************************************************************************
 *                              JTCOORDS                                     *
 *****************************************************************************}
procedure JTCOORDS(fn:S10; date:S8; var Fxy,Lxy:XYPtr; var nj,nxy:integer);
var  i,j:integer;  EH:boolean;  fk,ans:char;
{
    Reads and edits the joint coordinate data.
}
begin
   clrscr;        textcolor(6);
   gotoxy(9,1);   write(fn,'.DAT');
   gotoxy(32,1);  write('JOINT COORDINATES');
   gotoxy(64,1);  write(date);
   repeat
   EH:=False;
      SUBSXY(nxy);
      LISTXY(nxy,Fxy);
      if nxy=0  then for i:=1 to nj  do begin
         ADDXY(Fxy,Lxy,nxy);
         SUBSXY(nxy);
         LISTXY(nxy,Fxy)
      end;
      if nxy < nj then begin
         j:=nxy+1;
         for i:=j to nj  do begin
           repeat
              GTW24(29,22,'Add or Insert?  (A/I):  ');
              readln(ans);
              BLANK80(22);
              if (upcase(ans)<>'A') and (upcase(ans)<>'I')  then BEL
           until (upcase(ans)='A') or (upcase(ans)='I');
           if upcase(ans)='A'
              then ADDXY(Fxy,Lxy,nxy)
              else INXY(nxy,Fxy,Lxy);
```

```
            SUBSXY(nxy);
            LISTXY(nxy,Fxy)
         end
      end
      else begin
         if nxy > nj  then begin
            j:=nxy-nj;
            gotoxy(26,22);  write('You must delete ',j:1,' joint(s).');
            for i:=1 to j  do begin
               DELXY(Fxy,Lxy,nxy);
               SUBSXY(nxy);
               LISTXY(nxy,Fxy)
            end
         end
      end;
      BM;
      repeat
         read(kbd,fk);
         if (fk=#27) and keypressed  then begin
            EH:=True;
            read(kbd,fk);
            case fk of
               #59:CHANGEXY(Fxy,nj);
               #60:ADDXY(Fxy,Lxy,nxy);
               #61:INXY(nxy,Fxy,Lxy);
               #62:DELXY(Fxy,Lxy,nxy);
               #68:RTMM;
               else BMFK(6)
            end
         end
         else BMFK(6)
      until ((fk=#59) or (fk=#60) or (fk=#61) or (fk=#62) or (fk=#68))
            and EH;
   until (fk=#69) and EH and (nxy=nj)
end;



{*****************************************************************************
 *                                SUBST2                                     *
 *****************************************************************************}
procedure SUBST2(it,nc:integer);
var   H1:S12;   H2:S18;
{
   Writes the subheadings to the screen for the member properties.
}
begin
   BLANK80(4);    BLANK80(5);    BLANK80(24);    BLANK80(25);
   case it of
     1:begin     H1:='MEM    AREA ';     H2:='---  --------     '     end;
```

```
      2:begin     H1:='MEM      ZI ';     H2:='---  ----------   '     end;
      3:begin     H1:='MEM    EMOD ';     H2:='---  --------     '     end;
      4:begin     H1:='MEM     CTE ';     H2:='---  ---------    '     end
   end;
   if nc > 15  then begin
      if (it=1) or (it=3)  then begin
         GTW12(19,4,H1);  GTW18(19,5,H2);  GTW12(50,4,H1);  GTW18(50,5,H2)
      end
      else begin
         GTW12(18,4,H1);  GTW18(18,5,H2);  GTW12(49,4,H1);  GTW18(49,5,H2)
      end
   end
   else begin
      if it=2  then begin
         GTW12(33,4,H1);     GTW18(33,5,H2)
      end
      else begin
         GTW12(34,4,H1);     GTW18(34,5,H2)
      end
   end
end;


{*****************************************************************************
 *                                LIST2                                      *
 *****************************************************************************}
procedure LIST2(it,nc:integer; Ft2:T2Ptr);
var  i,j,j1:integer;  Tt2:T2Ptr;
{
   Lists the joint coordinates in a tabular form.
}
begin
   if nc=15  then begin
      for i:=6 to 20  do BLANK20(18,i);
      BLANK20(49,6)
   end;
   if nxy=16  then for i:=6 to 20  do BLANK20(33,i);
   Tt2:=Ft2;
   for i:=1 to nc  do begin
      if nc > 15
        then if i <= 15
               then if (it=1) or (it=3)
                      then gotoxy(19,(5+i))
                      else gotoxy(18,(5+i))
               else if (it=1) or (it=3)
                      then gotoxy(50,(i-10))
                      else gotoxy(49,(i-10))
        else if it=2
               then gotoxy(33,(5+i))
               else gotoxy(34,(5+i));
```

```
      with Tt2^  do begin
         case it of
            1:write(i:2,'  ',T21:8:3);
            2:write(i:2,'  ',T21:10:3);
            3:write(i:2,'  ',T21:8:1);
            4:write(i:2,'  ',T21:9:6)
         end
      end;
      Tt2:=Tt2^.next2
   end;
   if Ft2=nil
      then BLANK20(33,6)
      else if nc <= 15
         then BLANK20(33,i+6)
         else BLANK24(49,i-9);
end;


{*****************************************************************************
 *                                RANGET2                                    *
 *****************************************************************************}
procedure RANGET2(var Lt2:T2Ptr; it,ne:integer; var nc:integer);
var   i,j,k:integer;   p:real;   OK:boolean;
begin
   i:=nc+1;
   repeat
      gotoxy(23,22);   write('Enter range:  member ',i:2,' to member     ');
      gotoxy(58,22);
      {$I-}  readln(j);  {$I+}
      OK:=(IOresult=0);
      if (j < i) or (j > ne) or (not OK)  then BEL
   until (i <= j) and (j <= ne) and OK;
   repeat
      gotoxy(35,23);
      case it of
         1:begin
             write('A=              ');
             gotoxy(37,23)
           end;
         2:begin
             write('I=              ');
             gotoxy(37,23)
           end;
         3:begin
             write('E=              ');
             gotoxy(37,23)
           end;
         4:begin
             write('CTE=            ');
             gotoxy(39,23)
```

```
          end
       end;
       {$I-}  readln(p);  {$I+}
       OK:=(IOresult=0);
       if not OK  then BEL
    until OK;
    for k:=i to (j-1)  do begin
       Lt2^.T21:=p;
       new(Lt2^.next2);
       Lt2:=Lt2^.next2;
       nc:=nc+1
    end;
    Lt2^.T21:=p;
    Lt2^.next2:=nil;
    nc:=nc+1;
    BLANK80(22);     BLANK80(23)
end;


{*****************************************************************************
 *                                ADDT2                                      *
 *****************************************************************************}
procedure ADDT2(it:integer; var Ft2,Lt2:T2Ptr; var nc:integer);
var  j:integer;  OK:boolean;
{
    Adds a record of the type T2Ptr.
}
begin
    BMA;
    nc:=nc+1;
    repeat
       gotoxy(29,22);  write('Member ',nc:1,'   ');
       case it of
         1:begin
             write('A=              ');
             gotoxy(42,22)
           end;
         2:begin
             write('I=              ');
             gotoxy(42,22)
           end;
         3:begin
             write('E=              ');
             gotoxy(42,22)
           end;
         4:begin
             write('CTE=              ');
             gotoxy(44,22)
           end
       end;
```

```
      if Ft2=nil  then begin
         new(Ft2);
         Lt2:=Ft2
      end
      else begin
         new(Lt2^.next2);
         Lt2:=Lt2^.next2
      end;
      if it=4
         then gotoxy(44,22)
         else gotoxy(42,22);
      {$I-}  readln(Lt2^.T21);  {$I+}
      OK:=(IOresult=0);
      if not OK  then BEL
   until OK;
   Lt2^.next2:=nil;
   BLANK80(22);
end;


{*****************************************************************************
 *                                  DELT2                                    *
 *****************************************************************************}
procedure DELT2(var Ft2,Lt2:T2Ptr; it:integer; var nc:integer);
var  j,k,k1:integer;  ans:char;  Tt2,Nlt2:T2Ptr;  OK:boolean;
{
    Deletes a record of the type T2Ptr.
}
begin
   BMD;
   Tt2:=Ft2;
   repeat
      GTW25(29,23,'Delete member number=     ');
      gotoxy(50,23);
      {$I-}  readln(k); {$I+}
      OK:=(IOresult=0);
      if (k < 1) or (k > nc) or (not OK)  then BEL
   until (k >= 1) and (k <= nc) and OK;
   k1:=k-1;
   while k1 > 0  do begin
      k1:=k1-1;
      Nlt2:=Tt2;
      Tt2:=Tt2^.next2
   end;
   repeat
      for j:=22 to 23  do BLANK80(j);
      gotoxy(26,22);    write('Delete member ',k:1);
      with Tt2^  do begin
         case it of
           1:write(', A=',T21:8:3);
```

```
            2:write(', I=',T21:10:3);
            3:write(', E=',T21:8:1);
            4:write(', CTE=',T21:9:6)
         end
      end;
      write('?');
      GTW8(36,23,'(Y/N):  ');
      readln(ans);
      if (upcase(ans) <> 'Y') and (upcase(ans) <> 'N')  then BEL;
      if upcase(ans)='Y'  then begin
         if Ft2=Lt2  then begin
            dispose(Ft2);
            Ft2:=nil;
            Lt2:=Ft2
         end
         else begin
            for j:=k to (nc-1)  do begin
               if (k=1) and (j=1)
                  then Nlt2:=Tt2
                  else Nlt2:=Nlt2^.next2;
               Tt2:=Tt2^.next2;
               Nlt2^.T21:=Tt2^.T21;
            end;
            Lt2:=Nlt2;
            dispose(Lt2^.next2);
            Lt2^.next2:=nil
         end;
         nc:=nc-1
      end
   until (upcase(ans)='Y') or (upcase(ans)='N');
   for j:=22 to 23  do BLANK80(j)
end;


{*****************************************************************************
 *                              CHANGET2                                     *
 *****************************************************************************}
procedure CHANGET2(var Ft2:T2Ptr; it,nj:integer);
var  i:integer;  Tt2:T2Ptr;  x:real;  OK:boolean;
{
    Changes record of the type T2Ptr.
}
begin
   BMC;
   Tt2:=Ft2;
   repeat
      GTW25(29,22,'Change which member?      ');
      gotoxy(51,22);
      {$I-}  readln(i);  {$I+}
      OK:=(IOresult=0);
```

```
      if (i < 1) or (i > nj) or (not OK)  then BEL
   until (i >= 1) and (i <= nj) and OK;
   i:=i-1;
   while i > 0  do begin
      i:=i-1;
      Tt2:=Tt2^.next2
   end;
   with Tt2^  do begin
      repeat
         gotoxy(22,23);
         case it of
            1:write('Change A from ',T21:8:3,' to ');
            2:write('Change I from ',T21:10:3,' to ');
            3:write('Change E from ',T21:8:1,' to ');
            4:write('Change CTE from ',T21:9:6,' to ')
         end;
         {$I-}  readln(T21);  {$I+}
         OK:=(IOresult=0);
         if not OK  then BEL
      until OK
   end;
   for i:=22 to 23  do BLANK80(i)
end;


{*****************************************************************************
 *                                  INT2                                     *
 *****************************************************************************}
procedure INT2(it:integer; var nc:integer; var Ft2,Lt2:T2Ptr);
var  Tt2:T2Ptr;  i:integer;  w1,z1:real;  OK:boolean;
{
    Inserts a record of the type T2Ptr into the list of existing
    records.
}
begin
   BMI;
   new(Lt2^.next2);
   Lt2:=Lt2^.next2;
   Lt2^.next2:=nil;
   Tt2:=Ft2;
   repeat
      gotoxy(24,22);   write('Insert new member before member      ');
      gotoxy(56,22);
      {$I-}  readln(i);  {$I+}
      OK:=(IOresult=0);
      if (i < 1) or (i > nc) or (not OK)  then BEL
   until (i >= 1) and (i <= nc) and OK;
   i:=i-1;
   while i > 0  do begin
      i:=i-1;
```

```
      Tt2:=Tt2^.next2
   end;
   repeat
      gotoxy(35,23);
      case it of
         1:begin
             write('A=              ');
             gotoxy(37,23)
           end;
         2:begin
             write('I=              ');
             gotoxy(37,23)
           end;
         3:begin
             write('E=              ');
             gotoxy(37,23)
           end;
         4:begin
             write('CTE=              ');
             gotoxy(39,23)
           end
      end;
      {$I-} readln(w1); {$I+}
      OK:=(IOresult=0);
      if not OK then BEL
   until OK;
   while Tt2^.next2 <> nil do begin
      z1:=Tt2^.T21;
      Tt2^.T21:=w1;
      Tt2:=Tt2^.next2;
      w1:=z1;
   end;
   Tt2^.T21:=w1;
   nc:=nc+1;
   for i:=22 to 23 do BLANK80(i);
end;


{*****************************************************************************
 *                                TYPE2                                      *
 *****************************************************************************}
procedure TYPE2(var Ft2,Lt2:T2Ptr; it,ne:integer; var nc:integer);
var  i,j:integer;  EH:boolean;  fk,ans:char;
{
   Reads and edits the joint coordinate data.
}
begin
   for i:=8 to 12 do BLANK80(i);
   case it of
      1:GTW4(38,2,'AREA');
```

```
   2:GTW18(32,2,'MOMENT OF INERTIA ');
   3:GTW24(30,2,'MODULUS OF ELASTICITY   ');
   4:GTW36(25,2,'COEFFICIENT OF THERMAL EXPANSION    ')
end;
repeat
   EH:=False;
   SUBST2(it,nc);
   LIST2(it,nc,Ft2);
   if nc=0  then begin
      repeat
         if Ft2=nil  then begin
            new(Ft2);
            Lt2:=Ft2
         end
         else begin
            new(Lt2^.next2);
            Lt2:=Lt2^.next2
         end;
         RANGET2(Lt2,it,ne,nc);
         SUBST2(it,nc);
         LIST2(it,nc,Ft2)
      until nc=ne
   end;
   if nc < ne then begin
      repeat
         repeat
            GTW24(29,22,'Add or Insert?  (A/I):  ');
            readln(ans);
            BLANK80(22);
            if (upcase(ans)<>'A') and (upcase(ans)<>'I')  then BEL
         until (upcase(ans)='A') or (upcase(ans)='I');
         if upcase(ans)='A'  then begin
            new(Lt2^.next2);
            Lt2:=Lt2^.next2;
            RANGET2(Lt2,it,ne,nc);
            SUBST2(it,nc);
            LIST2(it,nc,Ft2)
         end
         else begin
            INT2(it,nc,Ft2,Lt2);
            SUBST2(it,nc);
            LIST2(it,nc,Ft2)
         end
      until nc=ne
   end
   else begin
      if nc > ne  then begin
         j:=nc-ne;
         gotoxy(26,22);  write('You must delete ',j:1,' member(s).');
         for i:=1 to j  do begin
```

```
            DELT2(Ft2,Lt2,it,nc);
            SUBST2(it,nc);
            LIST2(it,nc,Ft2)
          end
        end
      end;
      BM;
      repeat
        read(kbd,fk);
        if (fk=#27) and keypressed  then begin
          EH:=True;
          read(kbd,fk);
          case fk of
            #59:CHANGET2(Ft2,it,ne);
            #60:ADDT2(it,Ft2,Lt2,nc);
            #61:INT2(it,nc,Ft2,Lt2);
            #62:DELT2(Ft2,Lt2,it,nc);
            #68:begin
                  BLANK80(2);
                  for i:=4 to 20 do BLANK80(i);
                  BLANK80(24);
                  BLANK80(25)
                end;
            else BMFK(5)
          end
        end
        else BMFK(5)
    until ((fk=#59) or (fk=#60) or (fk=#61) or (fk=#62) or (fk=#68))
            and EH;
  until (fk=#68) and EH and (nc=ne)
end;


{*****************************************************************************
 *                                ELPROP                                     *
 *****************************************************************************}
procedure ELPROP(fn:S10; date:S8; var Fa,La,Fzi,Lzi,Fe,Le,Fcte,Lcte:T2Ptr;
                 var ne,na,nzi,nemod,ncte:integer);
var   fk:char;   EH:boolean;   i:integer;
{

   Reads and edits the element properties:  a, area, zi, moment of inertia
   about the local 3 axis, emod, the modulus of elasticity, and cte, the
   coefficient of thermal expansion.
}
  procedure EPFK;
  begin
    BEL;                textcolor(31);       GTW2(23,8,'F1');
    GTW2(23,9,'F2');    GTW2(23,10,'F3');    GTW2(23,11,'F4');
    GTW3(23,12,'F10');  textcolor(5);        gotoxy(1,25)
```

```
   end;

begin
   clrscr;        textcolor(5);
   gotoxy(9,1);   write(fn,'.DAT');
   gotoxy(32,1);  write('MEMBER PROPERTIES');
   gotoxy(64,1);  write(date);
   repeat
      GTW8(23,8,'F1 -AREA');
      GTW24(23,9,'F2 -MOMENT OF INERTIA   ');
      GTW25(23,10,'F3 -MODULUS OF ELASTICITY');
      GTW36(23,11,'F4 -COEFFICIENT OF THERMAL EXPANSION');
      GTW24(23,12,'F10-RETURN TO MAIN MENU ');
      gotoxy(1,25);
      EH:=False;
      repeat
         read(kbd,fk);
         if (fk=#27) and keypressed  then begin
            EH:=True;
            read(kbd,fk);
            case fk of
               #59:TYPE2(Fa,La,1,ne,na);          {A}
               #60:TYPE2(Fzi,Lzi,2,ne,nzi);       {I}
               #61:TYPE2(Fe,Le,3,ne,nemod);       {E}
               #62:TYPE2(Fcte,Lcte,4,ne,ncte);    {CTE}
               #68:begin
                     if (na <> ne) or (nzi <> ne) or (nemod <> ne) or
                        (ncte <> ne)  then begin
                        gotoxy(19,8);
                        write('There are ',ne:1,' members, you have only ',
                              'defined:');
                        for i:=9 to 12 do BLANK80(i);
                        if na <> ne  then begin
                           gotoxy(23,9);   write(na:2,'-Area')
                        end;
                        if nzi <> ne  then begin
                           gotoxy(23,10);
                           write(nzi:2,'-Moment of Inertia')
                        end;
                        if nemod <> ne  then begin
                           gotoxy(23,11);
                           write(nemod:2,'-Modulus of Elasticity')
                        end;
                        if ncte <> ne  then begin
                           gotoxy(23,12);
                           write(ncte:2,'-Coefficient of Thermal Expansion')
                        end;
                        delay(2000);
                        for i:=8 to 12 do BLANK80(i)
                     end
```

```
                end;
              else EPFK
            end
          end
          else EPFK
      until EH and ((fk=#59) or (fk=#60) or (fk=#61) or (fk=#62) or (fk=#68))
   until EH and (fk=#68) and (na=ne) and (nzi=ne) and (nemod=ne) and (ncte=ne);
   RTMM;
end;



{*****************************************************************************
 *                                  BM1                                      *
 *****************************************************************************}
{ Bottom Menu, One Line }
procedure BM1;
begin
   GTW6(14,25,'F1-ADD');        GTW9(36,25,'F2-DELETE');
   GTW10(58,25,'F10-RETURN');   gotoxy(1,25)
end;
procedure BM1FK; {Flashes BM1's function keys}
begin
   BEL;  textcolor(31);   GTW2(14,25,'F1');   GTW2(36,25,'F2');
   GTW3(58,25,'F10');     textcolor(11);      gotoxy(1,25)
end;
{ Load's Top Line }


{*****************************************************************************
 *                                 LOADSTL                                   *
 *****************************************************************************}
procedure LOADSTL(fn:S10; date:S8; lc:integer);
begin
   GTW8(9,1,fn);              write('.DAT');   GTW8(64,1,date);
   GTW10(35,1,'Load Case ');  write(lc:1)
end;


{*****************************************************************************
 *                                 SUBST3                                    *
 *****************************************************************************}
procedure SUBST3(j,nc:integer);
var  H1:S18;  H2:S24;
{
    Writes the subheadings to the screen for joint loads (j=1)
    or for prescribed joint displacements (j=2).
}
begin
   BLANK80(4);   BLANK80(5);   BLANK80(22);   BLANK80(23);   BLANK80(24);
```

```
   if j=1  then begin
      H1:='JT  DIR    FORCE  ';   H2:='--  ---  ----------     '
   end
   else begin
      H1:=' JT  DIR    DISP  ';   H2:=' --  ---  --------     '
   end;
   if nc > 15  then begin
      GTW18(15,4,H1);  GTW24(15,5,H2);
      GTW18(48,4,H1);  GTW24(48,5,H2)
   end
   else begin
      GTW18(31,4,H1);  GTW24(31,5,H2);
   end
end;


{*****************************************************************************
 *                                LIST3                                      *
 *****************************************************************************}
procedure LIST3(Ft3:T3Ptr; j,nc,t3s:integer);
var   i,ie,is,ix,iy,k:integer;   Tt3:T3Ptr;
{
    Lists the joint loads (j=1) or the prescribed joint displacements (j=2)
    in tabular form.
}
begin
   if nc=15  then begin
      for i:=6 to 20  do BLANK20(15,i);
      BLANK20(48,6)
   end;
   if nc=16  then for i:=6 to 20  do BLANK20(31,i);
   Tt3:=Ft3;
   for i:=1 to t3s  do Tt3:=Tt3^.next3;
   is:=nc div 30;
   ie:=nc mod 30;
   for i:=1 to is  do begin
      for k:=1 to 30  do begin
         if k <= 15  then begin
            if j=1
               then ix:=15
               else ix:=16;
            iy:=k+5
         end
         else begin
            if j=1
               then ix:=48
               else ix:=49;
            iy:=k-10
         end;
         BLANK20(ix,iy);
```

```
        gotoxy(ix,iy);
        with Tt3^  do
           if j=1
              then write(T31:2,'   ',T32:1,'   ',T33:10:3)
              else write(T31:2,'   ',T32:1,'   ',T33:8:3);
        Tt3:=Tt3^.next3
      end;
      GTW24(30,22,'Hit any key for more.   ');
      repeat  until keypressed;
      BLANK24(30,22)
    end;
    for i:=1 to ie  do begin
      if ie > 15  then begin
        if i <= 15  then begin
           if j=1
              then ix:=15
              else ix:=16;
           iy:=i+5
        end
        else begin
           if j=1
              then ix:=48
              else ix:=49;
           iy:=i-10
        end
      end
      else begin
        if j=1
           then ix:=31
           else ix:=32;
        iy:=i+5
      end;
      gotoxy(ix,iy);
      with Tt3^  do
        if j=1
           then write(T31:2,'   ',T32:1,'   ',T33:10:3)
           else write(T31:2,'   ',T32:1,'   ',T33:8:3);
      Tt3:=Tt3^.next3
    end;
    if nc=0
      then BLANK20(31,6)
      else BLANK20(ix,(iy+1));
    if is > 0  then
      for i:=(ie+1) to 30  do
        if i <= 15
          then BLANK20(31,(i+5))
          else BLANK20(48,(i-10))
end;
```

```
{*****************************************************************************
 *                                  PT3                                      *
 *****************************************************************************}
procedure PT3(var Ft3,Lt3:T3Ptr; j,jtno,jtdir,t3s:integer; var nc,tnc:integer;
            fordisp:real);
var   Tt3:T3Ptr;   i1,i2,j1,j2,isc:integer;   ri3,rj3:real;   c:char;
{
    Searches the linked list for the joint constraint which matches or is
    larger than the given joint constraint or the hinge which matches or is
    larger then the given hinge.  The new data is then inserted in the
    appropriate place.
}
begin
   Tt3:=Ft3;
   for i1:=1 to t3s  do Tt3:=Tt3^.next3;
   if Tt3=nil  then begin
      if Ft3=nil  then begin
         new(Ft3);
         Lt3:=Ft3
      end
      else begin
         new(Lt3^.next3);
         Lt3:=Lt3^.next3
      end;
      with Lt3^  do begin
         T31:=jtno;
         T32:=jtdir;
         T33:=fordisp;
         next3:=nil
      end
   end
   else begin
      isc:=1;
      while (isc <= nc) and (Tt3^.next3 <> nil) and (Tt3^.T31 <= jtno) and
         ((Tt3^.T31 <> jtno) or (Tt3^.T32 <= jtdir)) and
         ((Tt3^.T31 <> jtno) or (Tt3^.T32 <> jtdir) or (Tt3^.T33 <= fordisp))
         do begin
         isc:=isc+1;
         Tt3:=Tt3^.next3
      end;
      new(Lt3^.next3);
      Lt3:=Lt3^.next3;
      Lt3^.next3:=nil;
      if (isc > nc) or (Tt3^.T31 > jtno) or ((Tt3^.T31=jtno) and
         ((Tt3^.T32 > jtdir) or ((Tt3^.T32=jtdir) and (Tt3^.T33 > fordisp))))
         then begin
         i1:=jtno;
         i2:=jtdir;
         ri3:=fordisp;
         while Tt3^.next3 <> nil  do begin
```

```
            with Tt3^  do begin
               j1:=T31;
               j2:=T32;
               rj3:=T33;
               T31:=i1;
               T32:=i2;
               T33:=ri3
            end;
            Tt3:=Tt3^.next3;
            i1:=j1;
            i2:=j2;
            ri3:=rj3
         end;
         with Tt3^  do begin
            T31:=i1;
            T32:=i2;
            T33:=ri3
         end
      end
      else begin
         with Lt3^  do begin
            T31:=jtno;
            T32:=jtdir;
            T33:=fordisp
         end
      end
   end;
   nc:=nc+1;
   tnc:=tnc+1
end;


{*****************************************************************************
 *                                  ADDT3                                    *
 *****************************************************************************}
procedure ADDT3(var Ft3,Lt3:T3Ptr; j,nj,t3s:integer; var nc,tnc:integer);
var   jtno,jtdir,fjt,ljt:integer;  fk:char;   fordisp:real;   OK:boolean;
{
    Reads jtno, the joint to be loaded (j=1) or the joint with a prescribed
    displacement (j=2), jtdir, the direction of the joint load (j=1) or of
    the prescribed joint displacement (j=2), and fordisp, the force (j=1) or
    the displacement (j=2).  Calls PT3 to add to the linked list.
}
begin
   BLANK80(25);     GTW3(17,25,'Add');
   gotoxy(10,22);
   write('Choose S for single load or M for multiples of the same load.');
   repeat
      GTW18(32,23,'Enter S or M:     ');
      gotoxy(47,23);
```

```
   readln(fk);
   if (upcase(fk) <> 'S') and (upcase(fk) <> 'M')  then BEL
until (upcase(fk)='S') or (upcase(fk)='M');
BLANK80(22);   BLANK80(23);
if upcase(fk)='M'  then begin
   repeat
      GTW18(27,22,'First joint=      ');
      gotoxy(39,22);
      {$I-}  readln(fjt);  {$I+}
      OK:=(IOresult=0);
      if (fjt < 1) or (fjt > nj) or (not OK)  then BEL
   until (1 <= fjt) and (fjt <= nj) and OK;
   repeat
      GTW18(43,22,'Last joint=       ');
      gotoxy(54,22);
      {$I-}  readln(ljt);  {$I+}
      OK:=(IOresult=0);
      if (ljt < 1) or (ljt > nj) or (not OK)  then BEL
   until (1 <= ljt) and (ljt <= nj) and OK
end
else begin
   repeat
      GTW10(37,22,'Joint=    ');
      gotoxy(43,22);
      {$I-}  readln(fjt);  {$I+}
      OK:=(IOresult=0);
      if (fjt < 1) or (fjt > nj) or (not OK)
         then BEL
         else ljt:=fjt
   until (1 <= fjt) and (fjt <= nj) and OK
end;
repeat
   GTW18(20,23,'Direction=        ');
   gotoxy(30,23);
   {$I-}  readln(jtdir);  {$I+}
   OK:=(IOresult=0);
   if (jtdir < 1) or (jtdir > 3) or (not OK)  then BEL
until (1 <= jtdir) and (jtdir <= 3) and OK;
repeat
   if j=1  then begin
      GTW18(50,23,'Force=            ');
      gotoxy(56,23)
   end
   else begin
      GTW24(50,23,'Displacement=           ');
      gotoxy(63,23)
   end;
   {$I-}  readln(fordisp);  {$I+}
   OK:=(IOresult=0);
   if not OK  then BEL
```

```
   until OK;
   for jtno:=fjt to ljt  do PT3(Ft3,Lt3,j,jtno,jtdir,t3s,nc,tnc,fordisp)
end;


{*****************************************************************************
 *                                DELT3                                      *
 *****************************************************************************}
procedure DELT3(var Ft3,Lt3:T3Ptr; j,nj,t3s:integer; var nc,tnc:integer);
var  ic,ics,l,jtno,jtdir  :integer;
     Tt3,Nlt3             :T3Ptr;
     fordisp              :real;
     OK                   :boolean;
{
   Positions the linked list of joint loads (j=1) or prescribed joint
   displacements (j=2) at the proper record to be deleted then it
   shuffles the data down the records and disposes of the last record.
}
begin
   BLANK80(25);     GTW6(39,25,'Delete');
   ic:=0;
   Tt3:=Ft3;
   for l:=1 to t3s  do Tt3:=Tt3^.next3;
   repeat
      gotoxy(24,23);
      write('Enter joint number (1 to ',nj:1,'):       ');
      gotoxy(55,23);
      {$I-}  readln(jtno);  {$I+}
      OK:=(IOresult=0);
      if (jtno < 1) or (jtno > nj) or (not OK)  then BEL
   until (jtno >= 1) and (jtno <= nj) and OK;
   BLANK80(23);
   repeat
      gotoxy(22,23);
      write('Enter direction number (1 to 3):        ');
      gotoxy(56,23);
      {$I-}  readln(jtdir);  {$I+}
      OK:=(IOresult=0);
      if (jtdir < 1) or (jtdir > 3) or (not OK)  then BEL
   until (jtdir >= 1) and (jtdir <= 3) and OK;
   BLANK80(23);
   repeat
      if j=1  then begin
         GTW25(29,23,'Enter Force:            ');
         gotoxy(43,23)
      end
      else begin
         GTW36(26,23,'Enter Displacement:               ');
         gotoxy(47,23)
      end;
```

```
      {$I-} readln(fordisp); {$I+}
      OK:=(IOresult=0);
      if not OK then BEL
   until OK;
   ics:=1;
   while (Tt3^.T31 <= jtno) and (Tt3^.next3 <> nil) and (ics <= tnc) and
      ((Tt3^.T31 <> jtno) or (Tt3^.T32 <> jtdir) or (Tt3^.T33 <> fordisp))
      do begin
      ic:=ic+1;
      ics:=ics+1;
      Nlt3:=Tt3;
      Tt3:=Tt3^.next3
   end;
   if (Tt3^.T31=jtno) and (Tt3^.T32=jtdir) and (Tt3^.T33=fordisp) then begin
      if Lt3 <> Ft3 then begin
         ic:=ic+1;
         for l:=ic to (nc-1) do begin
            Nlt3:=Tt3;
            Tt3:=Tt3^.next3;
            Nlt3^.T31:=Tt3^.T31;
            Nlt3^.T32:=Tt3^.T32;
            Nlt3^.T33:=Tt3^.T33
         end;
         Lt3:=Nlt3;
         dispose(Lt3^.next3);
         Lt3^.next3:=nil
      end
      else begin
         dispose(Ft3);
         Ft3:=nil;
         Lt3:=Ft3
      end;
      nc:=nc-1;
      tnc:=tnc-1
   end
end;


{*****************************************************************************
 *                                TYPE3                                      *
 *****************************************************************************}
procedure TYPE3(fn:S10; date:S8; var Ft3,Lt3:T3Ptr; j,nj,lcn,t3s:integer;
              var nc,tnc:integer; var tpfac:real);
var   Tt3:T3Ptr;   i:integer;   fk:char;   EH,OK:boolean;
{
    Compiles and edits the joint load data (j=1) or the prescribed
    joint displacement data (j=2).  Calls SUBST3, LIST3, ADDT3,
    and DELT3.
}
begin
```

```
clrscr;  textcolor(11);  LOADSTL(fn,date,lcn);
if j=1  then begin
   gotoxy(35,2);  write('Joint Loads')
end
else begin
   gotoxy(26,2);  write('Prescribed Joint Displacements')
end;
if j=2  then begin
   if tpfac=0.0  then begin
      repeat
         GTW25(33,8,'PFAC=                    ');
         gotoxy(38,8);
         {$I-}  readln(tpfac);  {$I+}
         OK:=(IOresult=0);
         if not OK  then BEL
      until OK
   end
   else begin
      gotoxy(33,8);
      write('PFAC=',tpfac:7);
      repeat
         GTW24(30,9,'Change PFAC (Y/N)?      ');
         gotoxy(50,9);
         readln(fk);
         if (upcase(fk) <> 'Y') and (upcase(fk) <> 'N')  then BEL
      until (upcase(fk)='Y') or (upcase(fk)='N');
      if upcase(fk)='Y'  then begin
         repeat
            GTW25(33,10,'PFAC=                    ');
            gotoxy(38,10);
            {$I-}  readln(tpfac);  {$I+}
            OK:=(IOresult=0);
            if not OK  then BEL
         until OK
      end
   end;
   for i:=8 to 10  do BLANK80(i);
   gotoxy(35,3);
   write('PFAC=',tpfac:7)
end;
repeat
   BLANK80(23);
   BLANK80(24);
   EH:=False;
   SUBST3(j,nc);
   LIST3(Ft3,j,nc,t3s);
   BM1;
   repeat
      read(kbd,fk);
      if (fk=#27) and keypressed  then begin
```

```
            EH:=True;
            read(kbd,fk);
            case fk of
               #59:ADDT3(Ft3,Lt3,j,nj,t3s,nc,tnc);
               #60:DELT3(Ft3,Lt3,j,nj,t3s,nc,tnc);
               #68:begin  end;
               else BM1FK
            end
         end
         else BM1FK
      until ((fk=#59) or (fk=#60) or (fk=#68)) and EH;
   until (fk=#68) and EH
end;
```

```
{*****************************************************************************
 *                                SUBSMA                                     *
 *****************************************************************************}
procedure SUBSMA(nmact:integer);
const   H1='MEM MAT   FORCE     DIST1    DIST2  ';
        H2='--- --- ---------- -------- --------';
{
    Writes the subheadings to the screen for member actions.
}
begin
   BLANK80(4);   BLANK80(5);    BLANK80(22);   BLANK80(23);   BLANK80(24);
   if nmact > 15  then begin
      GTW36(3,4,H1);   GTW36(3,5,H2);
      GTW36(43,4,H1);  GTW36(43,5,H2)
   end
   else begin
      GTW36(23,4,H1);  GTW36(23,5,H2);
   end
end;
```

```
{*****************************************************************************
 *                                LISTMA                                     *
 *****************************************************************************}
procedure LISTMA(Fmact:MLPtr; nmact,mas:integer);
var   i,ie,is,ix,iy,k:integer;   Tmact:MLPtr;
{
   Lists the member actions in tabular form.
}
begin
   if (nmact=15) or (nmact=16)  then for i:=6 to 20  do BLANK80(i);
   Tmact:=Fmact;
```

```
   for i:=1 to mas  do Tmact:=Tmact^.nextml;
   is:=nmact div 30;
   ie:=nmact mod 30;
   for i:=1 to is  do begin
      for k:=1 to 30  do begin
         if k <= 15  then begin
            ix:=3;
            iy:=k+5
         end
         else begin
            ix:=43;
            iy:=k-10
         end;
         gotoxy(ix,iy);
         with Tmact^  do
            write(mln:2,'   ',mlt:1,'  ',mld:10:3,mld1:9:3,mld2:9:3);
         Tmact:=Tmact^.nextml
      end;
      GTW24(30,22,'Hit any key for more.   ');
      repeat  until keypressed;
      BLANK24(30,22)
   end;
   for i:=1 to ie  do begin
      if ie > 15  then begin
         if i <= 15
            then begin   ix:=3;    iy:=i+5    end
            else begin   ix:=43;   iy:=i-10   end
      end
      else begin   ix:=23;   iy:=i+5   end;
      gotoxy(ix,iy);
      with Tmact^  do
         write(mln:2,'   ',mlt:1,'  ',mld:10:3,mld1:9:3,mld2:9:3);
      Tmact:=Tmact^.nextml
   end;
   if nmact=0
      then BLANK40(23,6)
      else BLANK40(ix,(iy+1));
   if is > 0  then
      for i:=(ie+1) to 30  do
         if i <= 15
            then BLANK40(23,(i+5))
            else BLANK40(40,(i-10))
end;


{*****************************************************************************
 *                                  MATL                                     *
 *****************************************************************************}
procedure MATL(fn:S10; date:S8; lc:integer);
begin
```

```
   GTW8(9,1,fn);              write('.DAT');   GTW8(64,1,date);
   GTW10(35,1,'Load Case ');  write(lc:1)
end;


{*****************************************************************************
 *                                MAMENU                                     *
 *****************************************************************************}
procedure MAMENU(fn:S10; date:S8; lc:integer);
var   i,j,k,l:integer;

   procedure LMEM(i,j:integer); {Draws the little base line}
   begin
      draw(i,j,(i+159),j,1);
      draw(i,(j-2),i,(j+2),1);
      draw((i+159),(j-2),(i+159),(j+2),1)
   end;

begin
   clrscr;  hires;  hirescolor(11);  MATL(fn,date,lc);
   GTW24(31,2,'Adding Member Action   ');
   LMEM(80,42);
   draw(119,20,119,42,1);
   draw(115,24,119,20,1);
   draw(123,24,119,20,1);
   gotoxy(1,7);
   write('F1-Concentrated Lateral Load');
   LMEM(400,42);
   i:=419;
   while i <= 519  do begin
      draw(i,20,i,42,1);
      draw(i,20,(i-4),24,1);
      i:=i+20
   end;
   draw(419,20,519,20,1);
   gotoxy(41,7);
   write('F2-Uniformly Distributed Lateral Load');
   LMEM(80,75);
   draw(119,68,119,77,1);
   draw(84,68,119,68,1);
   draw(115,64,119,68,1);
   draw(115,72,119,68,1);
   gotoxy(1,12);
   write('F3-Concentrated Axial Load');
   LMEM(400,75);
   i:=400;
   while i <= 434  do begin
      draw(i,73,(i+10),73,1);
      draw((i+6),69,(i+10),73,1);
      i:=i+17
```

```
   end;
   while i <= 499  do begin
      draw(i,73,(i+9),73,1);
      draw((i+5),69,(i+9),73,1);
      i:=i+16
   end;
   while i <= 559  do begin
      draw(i,73,(i+10),73,1);
      draw((i+6),69,(i+10),73,1);
      i:=i+17
   end;
   GTW36(41,12,'F4-Uniformly Distributed Axial Load ');
   draw(146,100,135,123,1);
   draw(135,123,157,123,1);
   draw(157,123,146,100,1);
   draw(159,100,181,100,1);
   draw(170,100,170,123,1);
   GTW24(1,17,'F5-Temperature Change   ');
   LMEM(400,125);
   i:=494;
   while i <= 549  do begin
      draw(i,125,(i+6),125,0);
      i:=i+16
   end;
   i:=485;
   j:=125;
   for k:=1 to 3  do begin
      case k of
         1:l:=2;
         2:l:=3;
         3:l:=5
      end;
      draw(i,j,(i+28),(j-l),1);
      i:=i+28;
      j:=j-l
   end;
   GTW24(41,17,'F6-Element Imperfections');
   LMEM(240,162);
   draw(260,162,380,138,1);
   i:=290;
   j:=6;
   while i <= 380  do begin
      draw(i,162,i,(162-j),1);
      draw(i,(162-j),(i-4),(166-j),1);
      i:=i+30;
      j:=j+6
   end;
   gotoxy(20,22);
   write('F7-Triangularly Distributed Lateral Load')
end;
```

```
{*****************************************************************************
 *                                  BMEM                                     *
 *****************************************************************************}
procedure BMEM(j:integer); {Draws the big base line.}
var   i:integer;
begin
   draw(68,j,564,j,1);
   draw(68,(j-2),68,(j+2),1);
   draw(564,(j+2),564,(j-2),1);
   i:=(j div 8)+2;
   GTW1(9,i,'A');
   GTW1(71,i,'B')
end;


{*****************************************************************************
 *                                  DIML                                     *
 *****************************************************************************}
procedure DIML(i,j,k:integer); {Draws the dimension lines.}
begin
   draw(i,j,(i+k),j,1);
   draw((i-2),(j+2),(i+2),(j-2),1);
   draw((i+k-2),(j+2),(i+k+2),(j-2),1)
end;


{*****************************************************************************
 *                                  MA1                                      *
 *****************************************************************************}
procedure MA1(fn:S10; date:S8; lc:integer; var mtyp:integer; var force,
            dist1,dist2:real);
var   OK:boolean;
{
    Collects and shows data for member action type 1.
}
begin
   mtyp:=1;
   clrscr;
   hires;
   hirescolor(11);
   MATL(fn,date,lc);
   gotoxy(28,2);
   write('Concentrated Lateral Load');
   BMEM(132);
   draw(260,42,260,132,1);
   draw(260,42,256,46,1);
   draw(260,42,264,46,1);
   draw(68,148,68,157,1);
```

```
   draw(260,134,260,157,1);
   DIML(68,153,192);
   GTW1(33,5,'P');
   GTW4(20,19,'DIST');
   repeat
      GTW12(35,23,'P=          ');
      gotoxy(37,23);
      {$I-}  readln(force);  {$I+}
      OK:=(IOresult=0);
      if not OK  then BEL
   until OK;
   gotoxy(30,5);
   write(force:10:3,' k');
   repeat
      GTW18(34,24,'DIST=             ');
      gotoxy(39,24);
      {$I-}  readln(dist1);  {$I+}
      OK:=(IOresult=0);
      if (dist1 < 0) or (not OK)
         then BEL
         else dist2:=0.0
   until (dist1 >= 0) and OK;
   gotoxy(17,19);
   write(dist1:8:3,'"')
end;


{*****************************************************************************
 *                                  MA2                                      *
 *****************************************************************************}
procedure MA2(fn:S10; date:S8; lc:integer; var mtyp:integer; var force,
            dist1,dist2:real);
var   i:integer;   OK:boolean;
{
    Collects and shows data for member action type 2.
}
begin
   mtyp:=2;
   clrscr;
   hires;
   hirescolor(11);
   MATL(fn,date,lc);
   GTW36(23,2,' Uniformly Distributed Lateral Load ');
   BMEM(76);
   i:=156;
   while i <= 412  do begin
      draw(i,49,i,76,1);
      draw(i,49,(i-4),53,1);
      i:=i+64
   end;
```

```
   draw(156,49,412,49,1);
   draw(68,89,68,117,1);
   draw(156,78,156,101,1);
   draw(412,78,412,117,1);
   DIML(68,97,88);
   DIML(68,113,344);
   GTW1(36,6,'p');
   GTW6(12,12,'DIST1 ');
   GTW6(29,14,'DIST2 ');
   repeat
      GTW12(35,22,'p=          ');
      gotoxy(37,22);
      {$I-} readln(force); {$I+}
      OK:=(IOresult=0);
      if not OK  then BEL
   until OK;
   gotoxy(30,6);
   write(force:10:3,' kli');
   repeat
      GTW18(34,23,'DIST1=            ');
      gotoxy(40,23);
      {$I-} readln(dist1); {$I+}
      OK:=(IOresult=0);
      if (dist1 < 0) or (not OK)  then BEL
   until (dist1 >= 0) and OK;
   gotoxy(11,12);
   write(dist1:8:3,'"');
   repeat
      GTW18(34,24,'DIST2=            ');
      gotoxy(40,24);
      {$I-} readln(dist2); {$I+}
      OK:=(IOresult=0);
      if (dist2 <= dist1) or (not OK)  then BEL
   until (dist2 > dist1) and OK;
   gotoxy(27,14);
   write(dist2:8:3,'"')
end;


{*****************************************************************************
 *                                  MA3                                      *
 *****************************************************************************}
procedure MA3(fn:S10; date:S8; lc:integer; var mtyp:integer; var force,
            dist1,dist2:real);
var   OK:boolean;
{
    Collects and shows data for member action type 3.
}
begin
   mtyp:=3;
```

```
   clrscr;
   hires;
   hirescolor(11);
   MATL(fn,date,lc);
   GTW24(29,2,'Concentrated Axial Load ');
   BMEM(68);
   draw(444,68,444,58,1);
   draw(322,60,444,60,1);
   draw(444,60,440,56,1);
   draw(444,60,440,64,1);
   draw(68,81,68,93,1);
   draw(444,70,444,93,1);
   DIML(68,89,376);
   GTW1(40,8,'P');
   GTW4(32,11,'DIST');
   repeat
      GTW12(35,23,'P=          ');
      gotoxy(37,23);
      {$I-}  readln(force);  {$I+}
      OK:=(IOresult=0);
      if not OK  then BEL
   until OK;
   gotoxy(30,8);
   write(force:10:3,' k');
   GTW6(34,24,'DIST= ');
   repeat
      BLANK20(39,24);
      gotoxy(39,24);
      {$I-}  readln(dist1);  {$I+}
      OK:=(IOresult=0);
      if (dist1 < 0) or (not OK)
         then BEL
         else dist2:=0.0
   until (dist1 >= 0) and OK;
   gotoxy(29,11);
   write(dist1:8:3,'"')
end;


{*****************************************************************************
 *                                  MA4                                      *
 *****************************************************************************}
procedure MA4(fn:S10; date:S8; lc:integer; var mtyp:integer; var force,
            dist1,dist2:real);
var   i:integer;   OK:boolean;
{
    Collects and shows data for member action type 4.
}
begin
   mtyp:=4;
```

```
   clrscr;
   hires;
   hirescolor(11);
   MATL(fn,date,lc);
   GTW36(25,2,'Uniformaly Distributed Axial Load    ');
   BMEM(76);
   i:=68;
   while i <=132  do begin
      draw(i,74,(i+25),74,1);
      draw((i+25),74,(i+21),70,1);
      i:=i+32
   end;
   while i <= 443  do begin
      draw(i,74,(i+24),74,1);
      draw((i+24),74,(i+20),70,1);
      i:=i+31
   end;
   while i <= 564  do begin
      draw(i,74,(i+25),74,1);
      draw((i+25),74,(i+21),70,1);
      i:=i+32
   end;
   GTW1(40,9,'p');
   repeat
      GTW12(35,24,'p=          ');
      gotoxy(37,24);
      {$I-}  readln(force);  {$I+}
      OK:=(IOresult=0);
      if not OK  then BEL
   until OK;
   gotoxy(34,9);
   write(force:10:3,' kli');
   dist1:=0.0;
   dist2:=0.0
end;


{*****************************************************************************
 *                                  MA5                                      *
 *****************************************************************************}
procedure MA5(fn:S10; date:S8; lc:integer; var mtyp:integer; var force,
            dist1,dist2:real);
var   OK:boolean;
{
    Collects and shows data for member action type 7.
}
begin
   mtyp:=7;
   clrscr;
   hires;
```

```
   hirescolor(11);
   MATL(fn,date,lc);
   GTW18(32,2,'Temperature Change');
   draw(299,56,280,95,1);
   draw(299,56,318,95,1);
   draw(280,95,318,95,1);
   draw(339,56,339,95,1);
   draw(320,56,358,56,1);
   draw(267,184,264,190,1);
   draw(267,184,270,190,1);
   draw(264,190,270,190,1);
   repeat
      GTW12(35,24,'T=          ');
      gotoxy(37,24);
      {$I-}  readln(force);  {$I+}
      OK:=(IOresult=0);
      if not OK  then BEL
   until OK;
   dist1:=0.0;
   dist2:=0.0
end;


{*****************************************************************************
 *                                  MA6                                      *
 *****************************************************************************}
procedure MA6(fn:S10; date:S8; lc:integer; var mtyp:integer; var force,
            dist1,dist2:real);
var   i:integer;   OK:boolean;
{
    Collects and shows data for member action type 6.
}
begin
   mtyp:=6;
   clrscr;
   hires;
   hirescolor(11);
   MATL(fn,date,lc);
   GTW24(30,2,'Element Imperfections   ');
   BMEM(76);
   i:=353;
   while i <= 543  do begin
      draw(i,76,(i+20),76,0);
      i:=i+53
   end;
   draw(320,76,446,72,1);
   draw(446,72,573,68,1);
   draw(573,68,589,60,1);
   draw(591,59,623,43,1);
   draw(564,89,564,101,1);
```

```
   draw(589,62,589,101,1);
   draw(566,76,623,76,1);
   draw(591,60,623,60,1);
   DIML(564,97,25);
   draw(612,60,612,76,1);
   draw(614,58,610,62,1);
   draw(614,74,610,78,1);
   GTW2(72,12,'e1');
   GTW2(78,9,'e2');
   GTW2(78,7,'e3');
   repeat
      GTW18(34,22,'e1=              ');
      gotoxy(37,22);
      {$I-} readln(force); {$I+}
      OK:=(IOresult=0);
      if not OK  then BEL
   until OK;
   repeat
      GTW18(34,23,'e2=              ');
      gotoxy(37,23);
      {$I-} readln(dist1); {$I+}
      OK:=(IOresult=0);
      if not OK  then BEL
   until OK;
   repeat
      GTW18(34,24,'e3=              ');
      gotoxy(37,24);
      {$I-} readln(dist2); {$I+}
      OK:=(IOresult=0);
      if not OK  then BEL
   until OK
end;


{*****************************************************************************
 *                                  MA7                                      *
 *****************************************************************************}
procedure MA7(fn:S10; date:S8; lc:integer; var mtyp:integer; var force,
            dist1,dist2:real);
var   i,j:integer;   OK:boolean;
{
    Collects and shows data for member action type 5.
}
begin
   mtyp:=5;
   clrscr;
   hires;
   hirescolor(11);
   MATL(fn,date,lc);
   gotoxy(22,2);
```

```
   write('Triangularly Distributed Lateral Load');
   BMEM(76);
   i:=220;
   j:=6;
   while i <= 412  do begin
      draw(i,76,i,(76-j),1);
      draw(i,(76-j),(i-4),(80-j),1);
      i:=i+64;
      j:=j+7
   end;
   draw(156,76,412,49,1);
   draw(68,89,68,117,1);
   draw(156,78,156,99,1);
   draw(412,78,412,117,1);
   DIML(68,97,88);
   DIML(68,113,344);
   GTW1(52,6,'P');
   GTW6(12,12,'DIST1 ');
   GTW6(29,14,'DIST2 ');
   repeat
      GTW12(35,22,'P=          ');
      gotoxy(37,22);
      {$I-}  readln(force);  {$I+}
      OK:=(IOresult=0);
      if not OK  then BEL
   until OK;
   gotoxy(52,6);
   write(force:5:3,' k');
   GTW6(34,23,'DIST1=');
   repeat
      BLANK20(40,23);
      gotoxy(40,23);
      {$I-}  readln(dist1);  {$I+}
      OK:=(IOresult=0);
      if (dist1 < 0) or (not OK)  then BEL
   until (dist1 >= 0) and OK;
   gotoxy(10,12);
   write(dist1:8:3,'"');
   GTW6(34,24,'DIST2=');
   repeat
      BLANK20(40,24);
      gotoxy(40,24);
      {$I-}  readln(dist2);  {$I+}
      OK:=(IOresult=0);
      if (dist2 <= dist1) or (not OK)  then BEL
   until (dist2 > dist1) and OK;
   gotoxy(27,14);
   write(dist2:8:3,'"');
end;
```

```
{*****************************************************************************
 *                                  PMA                                      *
 *****************************************************************************}
procedure PMA(var Fmact,Lmact:MLPtr; mno,mtyp,mas:integer; var nmact,
            tnmact:integer; force,dist1,dist2:real);
var   Tmact:MLPtr;   i1,i2,j1,j2,isc:integer;   ri3,ri4,ri5,rj3,rj4,rj5:real;
      c:char;
{
    Searches the linked list for the member action type which matches or is
    larger than the given member action type.  The new data is then inserted
    in the appropriate place.
}
begin
   Tmact:=Fmact;
   for i1:=1 to mas  do Tmact:=Tmact^.nextml;
   if Tmact=nil  then begin
      if Fmact=nil  then begin
         new(Fmact);
         Lmact:=Fmact
      end
      else begin
         new(Lmact^.nextml);
         Lmact:=Lmact^.nextml
      end;
      with Lmact^  do begin
         mln:=mno;
         mlt:=mtyp;
         mld:=force;
         mld1:=dist1;
         mld2:=dist2;
         nextml:=nil
      end
   end
   else begin
      isc:=1;
      while (isc <= nmact) and (Tmact^.nextml <> nil) and (Tmact^.mln <= mno)
         and ((Tmact^.mln <> mno) or (Tmact^.mlt <= mtyp)) and
         ((Tmact^.mln <> mno) or (Tmact^.mlt <> mtyp) or (Tmact^.mld <= force))
         and ((Tmact^.mln <> mno) or (Tmact^.mlt <> mtyp) or
         (Tmact^.mld <> force) or (Tmact^.mld1 <= dist1)) and
         ((Tmact^.mln <> mno) or (Tmact^.mlt <> mtyp) or (Tmact^.mld <> force)
         or (Tmact^.mld1 <> dist1) or (Tmact^.mld2 <= dist2))  do begin
         isc:=isc+1;
         Tmact:=Tmact^.nextml
      end;
      new(Lmact^.nextml);
      Lmact:=Lmact^.nextml;
      Lmact^.nextml:=nil;
      if (isc > nmact) or (Tmact^.mln > mno) or ((Tmact^.mln=mno) and
```

```
        ((Tmact^.mlt > mtyp) or ((Tmact^.mlt=mtyp) and ((Tmact^.mld > force)
        or ((Tmact^.mld=force) and ((Tmact^.mld1 > dist1) or
        ((Tmact^.mld1=dist1) and (Tmact^.mld2 > dist2))))))))  then begin
        i1:=mno;
        i2:=mtyp;
        ri3:=force;
        ri4:=dist1;
        ri5:=dist2;
        while Tmact^.nextml <> nil  do begin
           with Tmact^  do begin
              j1:=mln;
              j2:=mlt;
              rj3:=mld;
              rj4:=mld1;
              rj5:=mld2;
              mln:=i1;
              mlt:=i2;
              mld:=ri3;
              mld1:=ri4;
              mld2:=ri5
           end;
           Tmact:=Tmact^.nextml;
           i1:=j1;
           i2:=j2;
           ri3:=rj3;
           ri4:=rj4;
           ri5:=rj5
        end;
        with Tmact^  do begin
           mln:=i1;
           mlt:=i2;
           mld:=ri3;
           mld1:=ri4;
           mld2:=ri5
        end
     end
     else begin
        with Lmact^  do begin
           mln:=mno;
           mlt:=mtyp;
           mld:=force;
           mld1:=dist1;
           mld2:=dist2
        end
     end
  end;
  nmact:=nmact+1;
  tnmact:=tnmact+1
end;
```

```
{*****************************************************************************
 *                                  ADDMA                                    *
 *****************************************************************************}
procedure ADDMA(var Fmact,Lmact:MLPtr; lcn,ne,mas:integer; var nmact,
                tnmact:integer);
var   mno,mtyp,fm,lm:integer;  fk:char;   force,dist1,dist2:real;   OK:boolean;
{
    Reads mno, the member to be loaded.  Calls MAi and PT3 to add to
    the linked list.
}
begin
   BLANK80(25);     GTW3(17,25,'Add');
   gotoxy(10,22);
   write('Choose S for single load or M for multiples of the same load.');
   repeat
      GTW18(32,23,'Enter S or M:     ');
      gotoxy(47,23);
      readln(fk);
      if (upcase(fk) <> 'S') and (upcase(fk) <> 'M')  then BEL
   until (upcase(fk)='S') or (upcase(fk)='M');
   BLANK80(22);   BLANK80(23);
   if upcase(fk)='M'  then begin
      repeat
         GTW18(26,22,'First member=      ');
         gotoxy(39,22);
         {$I-}  readln(fm);  {$I+}
         OK:=(IOresult=0);
         if (fm < 1) or (fm > ne) or (not OK)  then BEL
      until (1 <= fm) and (fm <= ne) and OK;
      repeat
         GTW18(43,22,'Last member=       ');
         gotoxy(55,22);
         {$I-}  readln(lm);  {$I+}
         OK:=(IOresult=0);
         if (lm < 1) or (lm > ne) or (not OK)  then BEL
      until (1 <= lm) and (lm <= ne) and OK
   end
   else begin
      repeat
         GTW12(36,22,'Member=     ');
         gotoxy(43,22);
         {$I-}  readln(fm);  {$I+}
         OK:=(IOresult=0);
         if (fm < 1) or (fm > ne) or (not OK)
            then BEL
           else lm:=fm
      until (1 <= fm) and (fm <= ne) and OK
   end;
   EH:=False;
```

```
   MAMENU(fn,date,lcn);
   gotoxy(1,25);
   repeat
      read(kbd,fk);
      if (fk=#27) and keypressed  then begin
         EH:=True;
         read(kbd,fk);
         case fk of
            #59:MA1(fn,date,lcn,mtyp,force,dist1,dist2);
            #60:MA2(fn,date,lcn,mtyp,force,dist1,dist2);
            #61:MA3(fn,date,lcn,mtyp,force,dist1,dist2);
            #62:MA4(fn,date,lcn,mtyp,force,dist1,dist2);
            #63:MA5(fn,date,lcn,mtyp,force,dist1,dist2);
            #64:MA6(fn,date,lcn,mtyp,force,dist1,dist2);
            #65:MA7(fn,date,lcn,mtyp,force,dist1,dist2);
            else BEL
         end
      end
      else BEL
   until EH and ((fk=#59) or (fk=#60) or (fk=#61) or (fk=#62) or (fk=#63)
                 or (fk=#64) or (fk=#65));
   for mno:=fm to lm  do
      PMA(Fmact,Lmact,mno,mtyp,mas,nmact,tnmact,force,dist1,dist2);
   GTW24(29,25,'Hit any key to continue.');
   repeat  until keypressed;
   textmode
end;


(*****************************************************************************
 *                                  DELMA                                    *
 *****************************************************************************)
procedure DELMA(var Fmact,Lmact:MLPtr; ne,mas:integer; var nmact,
                tnmact:integer);
var   ic,ics,l,mno,mtyp  :integer;
      Tmact,Nlmact       :MLPtr;
      force,dist1,dist2  :real;
      OK                 :boolean;
{
   Positions the linked list of member actions at the proper record to
   be deleted then it shuffles the data down the records and disposes
   of the last record.
}
begin
   BLANK80(25);     GTW6(39,25,'Delete');
   ic:=0;
   Tmact:=Fmact;
   for l:=1 to mas  do Tmact:=Tmact^.nextml;
   repeat
      gotoxy(24,23);
```

```
   write('Enter member number (1 to ',ne:1,'):        ');
   gotoxy(56,23);
   {$I-} readln(mno); {$I+}
   OK:=(IOresult=0);
   if (mno < 1) or (mno > ne) or (not OK) then BEL
until (mno >= 1) and (mno <= ne) and OK;
BLANK80(23);
repeat
   gotoxy(21,23); write('Enter member action type (1 to 7):        ');
   gotoxy(57,23);
   {$I-} readln(mtyp); {$I+}
   OK:=(IOresult=0);
   if (mtyp < 1) or (mtyp > 7) or (not OK) then BEL
until (mtyp >= 1) and (mtyp <= 7) and OK;
BLANK80(23);
repeat
   GTW24(29,23,'Enter Force:           ');
   gotoxy(43,23);
   {$I-} readln(force); {$I+}
   OK:=(IOresult=0);
   if not OK then BEL
until OK;
BLANK80(23);
repeat
   GTW36(28,23,'Enter Distance1:                  ');
   gotoxy(46,23);
   {$I-} readln(dist1); {$I+}
   OK:=(IOresult=0);
   if not OK then BEL
until OK;
BLANK80(23);
repeat
   GTW36(28,23,'Enter Distance2:                  ');
   gotoxy(46,23);
   {$I-} readln(dist2); {$I+}
   OK:=(IOresult=0);
   if not OK then BEL
until OK;
BLANK80(23);
ics:=1;
while (Tmact^.mln <= mno) and (Tmact^.nextml <> nil) and (ics <= tmmact) and
   ((Tmact^.mln <> mno) or (Tmact^.mlt <> mtyp) or (Tmact^.mld <> force) or
   (Tmact^.mld1 <> dist1) or (Tmact^.mld2 <> dist2)) do begin
   ic:=ic+1;
   ics:=ics+1;
   Nlmact:=Tmact;
   Tmact:=Tmact^.nextml
end;
if (Tmact^.mln=mno) and (Tmact^.mlt=mtyp) and (Tmact^.mld=force) and
   (Tmact^.mld1=dist1) and (Tmact^.mld2=dist2) then begin
```

```
      if Lmact <> Fmact  then begin
         ic:=ic+1;
         for i:=ic to (nmact-1)  do begin
            Nlmact:=Tmact;
            Tmact:=Tmact^.nextml;
            Nlmact^.mln:=Tmact^.mln;
            Nlmact^.mlt:=Tmact^.mlt;
            Nlmact^.mld:=Tmact^.mld;
            Nlmact^.mld1:=Tmact^.mld1;
            Nlmact^.mld2:=Tmact^.mld2
         end;
         Lmact:=Nlmact;
         dispose(Lmact^.nextml);
         Lmact^.nextml:=nil
      end
      else begin
         dispose(Fmact);
         Fmact:=nil;
         Lmact:=Fmact
      end;
      nmact:=nmact-1;
      tnmact:=tnmact-1
   end
end;


{*****************************************************************************
 *                                  MACT                                     *
 *****************************************************************************}
procedure MACT(fn:S10; date:S8; var Fmact,Lmact:MLPtr; ne,lcn,mas:integer;
               var nmact,tnmact:integer);
var   Tmact:MLPtr;   i:integer;   fk:char;   EH:boolean;
{
    Compiles and edits the member action data.  Calls SUBSMA, LISMA,
    ADDMA, and DELMA.
}
begin
   repeat
      clrscr;         textcolor(11);            LOADSTL(fn,date,lcn);
      gotoxy(34,2);   write('Member Actions');   BLANK80(23);
      BLANK80(24);    EH:=False;
      SUBSMA(nmact);
      LISTMA(Fmact,nmact,mas);
      BM1;
      repeat
         read(kbd,fk);
         if (fk=#27) and keypressed  then begin
            EH:=True;
            read(kbd,fk);
            case fk of
```

```
            #59:ADDMA(Fmact,Lmact,lcm,ne,mas,nmact,tmact);
            #60:DELMA(Fmact,Lmact,ne,mas,nmact,tmact);
            #68:begin  end;
            else BMIFK
          end
        end
        else BMIFK
     until ((fk=#59) or (fk=#60) or (fk=#68)) and EH
  until (fk=#68) and EH
end;
{*****************************************************************************
 *                                LOADTL                                     *
 *****************************************************************************}
procedure LOADTL(fn:S10; date:S8);
{
   Writes the top line for LOADS to the screen.
}
begin
  clrscr;       textcolor(11);   gotoxy(9,1);   write(fn:10,'.DAT');
  gotoxy(64,1); write(date:8);   gotoxy(36,1);  write('LOAD DATA');
  gotoxy(1,25);
end;


{*****************************************************************************
 *                                DELNVLR                                    *
 *****************************************************************************}
procedure DELNVLR(Flccv:ControlPtr; var Fjl,Ljl,Fpjd,Lpjd:T3Ptr;
                  var Fmact,Lmact:MLPtr; ne,nj,nlccv:integer;
                  var tnjl,tnmact,tnpjd:integer);
var   Tlccv,Nllccv:ControlPtr;   Tjl,Nljl,Tpjd,Nlpjd,S1t3,S2t3:T3Ptr;
      Tmact,Nlmact,S1ml,S2ml:MLPtr;  i,j,k:integer;
{
   Find and dispose of any load records which are no longer
    valid due to a change of NJ or NE.
}
begin
  Tlccv:=Flccv;   Tjl:=Fjl;   Tmact:=Fmact;   Tpjd:=Fpjd;
  for i:=1 to nlccv  do begin
    if Tlccv^.connjl > 0  then begin
       j:=1;
       while (Tjl^.T31 <= nj) and (j <= Tlccv^.connjl)  do begin
          Nljl:=Tjl;
          Tjl:=Tjl^.next3;
          j:=j+1
       end;
       if (Tjl^.T31 > nj) and (j <= Tlccv^.connjl)  then begin
          for k:=j to Tlccv^.connjl  do begin
             if Tjl^.next3=nil  then begin
                if Fjl=Ljl  then begin
```

```
                dispose(Fjl);
                Fjl:=nil;
                Ljl:=Fjl
             end
             else begin
                Ljl:=Nljl;
                dispose(Ljl^.next3);
                Ljl^.next3:=nil
             end;
             Tjl:=nil
          end
          else begin
             S1t3:=Tjl;
             repeat
                S2t3:=S1t3;
                S1t3:=S1t3^.next3;
                S2t3^.T31:=S1t3^.T31;
                S2t3^.T32:=S1t3^.T32;
                S2t3^.T33:=S1t3^.T33
             until S1t3^.next3=nil;
             Ljl:=S2t3;
             dispose(Ljl^.next3);
             Ljl^.next3:=nil
          end;
          tmjl:=tmjl-1;
          Tlccv^.connjl:=Tlccv^.connjl-1
       end
    end
end;
if Tlccv^.conmat > 0  then begin
    j:=1;
    while (Tmact^.mln <= ne) and (j <= Tlccv^.conmat)  do begin
       Nlmact:=Tmact;
       Tmact:=Tmact^.nextml;
       j:=j+1
    end;
    if (Tmact^.mln > ne) and (j <= Tlccv^.conmat)  then begin
       for k:=j to Tlccv^.conmat  do begin
          if Tmact^.nextml=nil  then begin
             if Fmact=Lmact  then begin
                dispose(Fmact);
                Fmact:=nil;
                Lmact:=Fmact
             end
             else begin
                Lmact:=Nlmact;
                dispose(Lmact^.nextml);
                Lmact^.nextml:=nil
             end;
             Tmact:=nil
```

```
        end
        else begin
           S1ml:=Tmact;
           repeat
              S2ml:=S1ml;
              S1ml:=S1ml^.nextml;
              S2ml^.mln:=S1ml^.mln;
              S2ml^.mlt:=S1ml^.mlt;
              S2ml^.mld:=S1ml^.mld;
              S2ml^.mld1:=S1ml^.mld1;
              S2ml^.mld2:=S1ml^.mld2
           until S1ml^.nextml=nil;
           Lmact:=S2ml;
           dispose(Lmact^.nextml);
           Lmact^.nextml:=nil
        end;
        tmmact:=tmmact-1;
        Tlccv^.conmat:=Tlccv^.conmat-1
     end
  end
end;
if Tlccv^.connpjd > 0  then begin
   j:=1;
   while (Tpjd^.T31 <= nj) and (j <= Tlccv^.connpjd)  do begin
      Nlpjd:=Tpjd;
      Tpjd:=Tpjd^.next3;
      j:=j+1
   end;
   if (Tpjd^.T31 > nj) and (j <= Tlccv^.connpjd)  then begin
      for k:=j to Tlccv^.connpjd  do begin
         if Tpjd^.next3=nil  then begin
            if Fpjd=Lpjd  then begin
               dispose(Fpjd);
               Fpjd:=nil;
               Lpjd:=Fpjd
            end
            else begin
               Lpjd:=Nlpjd;
               dispose(Lpjd^.next3);
               Lpjd^.next3:=nil
            end;
            Tpjd:=nil
         end
         else begin
            S1t3:=Tpjd;
            repeat
               S2t3:=S1t3;
               S1t3:=S1t3^.next3;
               S2t3^.T31:=S1t3^.T31;
               S2t3^.T32:=S1t3^.T32;
```

```
                S2t3^.T33:=S1t3^.T33
              until S1t3^.next3=nil;
              Lpjd:=S2t3;
              dispose(Lpjd^.next3);
              Lpjd^.next3:=nil
            end;
            tnpjd:=tnpjd-1;
            Tlccv^.connpjd:=Tlccv^.connpjd-1
          end
        end
      end;
      Tlccv:=Tlccv^.nextcon
   end
end;


{*****************************************************************************
 *                                DELNVLC                                    *
 *****************************************************************************}
procedure DELNVLC(var Flccv,Llccv:ControlPtr; var Fjl,Ljl,Fpjd,Lpjd:T3Ptr;
                  var Fmact,Lmact:MLPtr; var fk:char; nlc:integer;
                  var nlccv,tnjl,tnmact,tnpjd:integer);
var   i,j,k,jls,mas,pjds   :integer;
      Tlccv,N1lccv         :ControlPtr;
      Tmact,N1mact         :MLPtr;
      Tjl,N1jl,Tpjd,N1pjd  :T3Ptr;
      OK                   :boolean;
{
   Delete load cases that are no longer valid due to change in NLC, or
   return to MAIN MENU to change NLC to a larger number.
}
begin
   while nlccv > nlc  do begin
      gotoxy(13,9);
      write('You have set NLC=',nlc:1,' but there are ',nlccv:1,
            ' load cases in memory.');
      gotoxy(8,10);
      write('You may either Return to MAIN to change NLC or Delete ',
            'a load case.');
      repeat
         GTW46(22,11,'Enter R to Return or D to Delete:           ');
         gotoxy(58,11);
         readln(fk);
         if (upcase(fk) <> 'R') or (upcase(fk) <> 'D')  then BEL
      until (upcase(fk)='R') or (upcase(fk)='D');
      if upcase(fk)='R'  then exit;
      repeat
         GTW36(28,12,'Delete which load case?            ');
         gotoxy(53,12);
         {$I-}  readln(i);  {$I+}
```

```
   OK:=(IOresult=0);
   if (i < 1) or (i > nlccv) or (not OK)  then BEL
until (i >= 1) and (i <= nlccv) and OK;
Tlccv:=Flccv;
jls:=0;   mas:=0;   pjds:=0;
for j:=1 to (i-1)  do begin
   with Tlccv^  do begin
      jls:=jls+connjl;
      mas:=mas+connmat;
      pjds:=pjds+connpjd
   end;
   Tlccv:=Tlccv^.nextcon
end;
for j:=10 to 12  do BLANK80(j);
gotoxy(22,11);
with Tlcc
   do write('Load Case ',i:1,':  NJL=',connjl:1,', NMACT=',conmmat:1,
            ', NPJD=',connpjd:1);
repeat
   gotoxy(15,12);
   write('Enter R to Return or D to Delete Load Case ',i:1,':     ');
   gotoxy(63,12);
   readln(fk);
   if (upcase(fk) <> 'R') and (upcase(fk) <> 'D')  then BEL
until (upcase(fk)='R') or (upcase(fk)='D');
if upcase(fk)='R'  then exit;
for j:=(jls+1) to (jls+Tlccv^.connjl)  do begin
   Tjl:=Fjl;
   Nljl:=Tjl;
   for k:=1 to jls  do Tjl:=Tjl^.next3;
   if Ljl <> Fjl  then begin
      for k:=(jls+2) to tnjl  do begin
         Nljl:=Tjl;
         Tjl:=Tjl^.next3;
         Nljl^.T31:=Tjl^.T31;
         Nljl^.T32:=Tjl^.T32;
         Nljl^.T33:=Tjl^.T33
      end;
      Ljl:=Nljl;
      dispose(Ljl^.next3);
      Ljl^.next3:=nil
   end
   else begin
      dispose (Fjl);
      Fjl:=nil;
      Ljl:=Fjl
   end;
   tnjl:=tnjl-1
end;
for j:=(mas+1) to (mas+Tlccv^.conmmat)  do begin
```

```
    Tmact:=Fmact;
    Nlmact:=Tmact;
    for k:=1 to mas  do Tmact:=Tmact^.nextml;
    if Lmact <> Fmact  then begin
       for k:=(mas+2) to tnmact  do begin
          Nlmact:=Tmact;
          Tmact:=Tmact^.nextml;
          Nlmact^.mln:=Tmact^.mln;
          Nlmact^.mlt:=Tmact^.mlt;
          Nlmact^.mld:=Tmact^.mld;
          Nlmact^.mld1:=Tmact^.mld1;
          Nlmact^.mld2:=Tmact^.mld2
       end;
       Lmact:=Nlmact;
       dispose(Lmact^.nextml);
       Lmact^.nextml:=nil
    end
    else begin
       dispose (Fmact);
       Fmact:=nil;
       Lmact:=Fmact
    end;
    tnmact:=tnmact-1
end;
for j:=(pjds+1) to (pjds+Tlccv^.connpjd)  do begin
    Tpjd:=Fpjd;
    Nlpjd:=Tpjd;
    for k:=1 to pjds  do Tpjd:=Tpjd^.next3;
    if Lpjd <> Fpjd  then begin
       for k:=(pjds+2) to tnpjd  do begin
          Nlpjd:=Tpjd;
          Tpjd:=Tpjd^.next3;
          Nlpjd^.T31:=Tpjd^.T31;
          Nlpjd^.T32:=Tpjd^.T32;
          Nlpjd^.T33:=Tpjd^.T33
       end;
       Lpjd:=Nlpjd;
       dispose(Lpjd^.next3);
       Lpjd^.next3:=nil
    end
    else begin
       dispose (Fpjd);
       Fpjd:=nil;
       Lpjd:=Fpjd
    end;
    tnpjd:=tnpjd-1
end;
Nllccv:=Tlccv;
for j:=1 to (nlccv-1)  do begin
    Nllccv:=Tlccv;
```

```
        Tlccv:=Tlccv^.nextcon;
        Nllccv^.connjl:=Tlccv^.connjl;
        Nllccv^.connmat:=Tlccv^.connmat;
        Nllccv^.connpjd:=Tlccv^.connpjd;
        Nllccv^.conpfac:=Tlccv^.conpfac
      end;
      Llccv:=Nllccv;
      dispose(Llccv^.nextcon);
      Llccv^.nextcon:=nil;
      nlccv:=nlccv-1
   end
end;


(*****************************************************************************
 *                                  LOADS                                    *
 *****************************************************************************)
procedure LOADS(fn:S10; date:S8; var Fjl,Ljl,Fpjd,Lpjd:T3Ptr;
                var Fmact,Lmact:MLPtr; var Flccv,LLccv:ControlPtr;
                ne,nj,nlc:integer; var tnjl,tnmact,tnpjd,nlccv:integer);
var  EH,OK:boolean;

     fk                   :char;
     dr                   :real;
     lcn,j,k,jls,mas,pjds :integer;
     Tlccv,Nllccv         :ControlPtr;
     Tjl,Nljl,Tpjd,Nlpjd  :T3Ptr;
     Tmact,Nlmact         :MLPtr;
{
    Calls DELNVLR, DELNVLC, JTLOADS, MEMLOADS, and PJDISPS.
}

   { Flash F keys }
   procedure LFK;
   begin
      textcolor(31);      BEL;           GTW2(31,9,'F1');
      GTW3(31,10,'F10');  gotoxy(1,25);  textcolor(11)
   end;
   procedure LTFK;
   begin
      textcolor(31);      BEL;               GTW2(24,8,'F1');
      GTW2(24,9,'F2');    GTW2(24,10,'F3');  GTW3(24,11,'F10');
      gotoxy(1,25);       textcolor(11)
   end;


begin
   DELNVLR(Flccv,Fjl,Ljl,Fpjd,Lpjd,Fmact,Lmact,ne,nj,nlccv,tnjl,tnmact,tnpjd);
   LOADTL(fn,date);
   DELNVLC(Flccv,Llccv,Fjl,Ljl,Fpjd,Lpjd,Fmact,Lmact,fk,nlc,nlccv,tnjl,
```

```
        tmmact,tmpjd);
if upcase(fk)='R'  then begin
   RTMM;
   exit
end;
repeat
   LOADTL(fn,date);
   EH:=False;
   lcn:=nlccv+1;
   if (nlc=nlccv) and (nlc > 1)  then begin
      GTW24(31,9,'F1 -SELECT LOAD CASE    ');
      GTW10(31,10,'F10-RETURN');
      gotoxy(1,25);
      repeat
         read(kbd,fk);
         if (fk=#27) and keypressed  then begin
            EH:=True;
            read(kbd,fk);
            case fk of
               #59:begin
                     repeat
                        BLANK80(10);
                        gotoxy(26,16);
                        write('Edit load case (1 to ',nlc:1,');     ');
                        gotoxy(53,16);
                        {$I-}  readln(lcn);  {$I+}
                        OK:=(IOresult=0);
                        if (lcn < 1) or (lcn > nlc) or (not OK)  then BEL
                     until (1 <= lcn) and (lcn <= nlc) and OK;
                  end;
               #68:lcn:=0;
               else LFK
            end
         end
         else LFK
      until EH and ((fk=#59) or (fk=#68));
      BLANK80(9);   BLANK80(10);   BLANK80(16)
   end
   else if (nlc=nlccv) and (nlc=1)  then lcn:=1;
   jls:=0;   mas:=0;   pjds:=0;   Tlccv:=Flccv;   dr:=0.0;
   for j:=1 to (lcn-1)  do begin
      with Tlccv^  do begin
         jls:=jls+connjl;
         mas:=mas+connmat;
         pjds:=pjds+connpjd
      end;
      Tlccv:=Tlccv^.nextcon
   end;
   if nlccv < nlc  then begin
      if Flccv=nil  then begin
```

```
      new(Flccv);
      Llccv:=Flccv;
   end
   else begin
      new(Llccv^.nextcon);
      Llccv:=Llccv^.nextcon;
   end;
   with Llccv^  do begin
      connjl:=0;
      connmat:=0;
      connpjd:=0;
      conpfac:=0.0;
      nextcon:=nil
   end;
   Tlccv:=Llccv
end;
if lcn <> 0  then begin
   repeat
      LOADTL(fn,date);
      gotoxy(35,1);
      write('Load Case ',lcn:1);
      GTW4(38,2,'Menu');
      GTW18(24,8,'F1 -Joint Loads   ');
      GTW18(24,9,'F2 -Member Actions');
      GTW36(24,10,'F3 -Prescribed Joint Displacements  ');
      GTW18(24,11,'F10-End Load Case ');
      gotoxy(1,25);
      EH:=False;
      repeat
         read(kbd,fk);
         if (fk=#27) and keypressed  then begin
            EH:=True;
            read(kbd,fk);
            case fk of
               #59:TYPE3(fn,date,Fjl,Ljl,1,nj,lcn,jls,Tlccv^.connjl,
                         tnjl,dr);
               #60:MACT(fn,date,Fmact,Lmact,ne,lcn,mas,Tlccv^.conmmat,
                        tnmact);
               #61:TYPE3(fn,date,Fpjd,Lpjd,2,nj,lcn,pjds,Tlccv^.connpjd,
                         tnpjd,Tlccv^.conpfac);
               #68:begin
                     for k:=22 to 24  do BLANK80(k);
                     if nlccv < nlc  then begin
                        with Tlccv^  do begin
                           if (connjl=0) and (connmat=0) and (connpjd=0)
                              then begin
                              gotoxy(19,23);
                              write('You have created a load case ',
                                    'with no loads.   ');
                              delay(2500)
```

```
                              end
                            end;
                            nlccv:=nlccv+1;
                            if (nlccv=nlc) and (nlc=1)  then lcn:=0;
                          end
                          else if nlc=1  then lcn:=0
                        end;
                    else LTFK
                  end
                end
                else LTFK
            until EH and ((fk=#59) or (fk=#60) or (fk=#61) or (fk=#68))
          until EH and (fk=#68)
        end
    until (lcn=0) and (nlccv=nlc);
    RTMM
end;



(*****************************************************************************
 *                                DRAWS                                      *
 *****************************************************************************)
procedure DRAWS(fn,date:S8; Fxy,Lxy:XYPtr; Fmi,Lmi:MIPtr; Fh,Lh:TIPtr;
                ne,nj,nh:integer);
type
   Icoords=array[1..30] of integer;
var
   CPtr     :XYPtr;
   MPtr     :MIPtr;
   HPtr     :TIPtr;
   ix,iy    :Icoords;
   i,j,k,l  :integer;
   minx,miny,maxx,maxy,xrange,yrange,sfx,sfy,xc,yc  :real;

{
    Uses the joint coordinates, member incidences, and the hinge data
    to graphically depict the frame model on the screen.
}
begin
   clrscr;          textcolor(1);
   clrscr;           hires;              hirescolor(1);
   gotoxy(1,1);      write(fn,'.DAT');
   gotoxy(3,2);      write(date);
   minx:=Fxy^.x1;    miny:=Fxy^.x2;
   maxx:=Fxy^.x1;    maxy:=Fxy^.x2;
   CPtr:=Fxy;
   for i:=1 to nj  do begin
      with CPtr^  do begin
         if x1 < minx  then minx:=x1;
```

```
      if x1 > maxx  then maxx:=x1;
      if x2 < miny  then miny:=x2;
      if x2 > maxy  then maxy:=x2
   end;
   if j <> nj  then CPtr:=CPtr^.nextx
end;
xrange:=maxx-minx;
yrange:=maxy-miny;
if xrange >= yrange  then begin
   sfx:=1.0;
   sfy:=yrange/xrange
end
else begin
   sfy:=1.0;
   sfx:=xrange/yrange
end;
xc:=minx+0.5*xrange;
yc:=miny+0.5*yrange;
CPtr:=Fxy;
for j:=1 to nj  do begin
   with CPtr^  do begin
      if xc <> 0
         then ix[j]:=round(329+209*sfx*(x1-xc)/xc)
         else ix[j]:=421;
      if yc <> 0
         then iy[j]:=round(99-90*sfy*(x2-yc)/yc)
         else iy[j]:=99
   end;
   if j <> nj  then Cptr:=Cptr^.nextx
end;
for j:=1 to nj  do begin
   plot(ix[j],iy[j],1);
   draw(ix[j]+1,iy[j]+1,ix[j]+1,iy[j]-1,1);
   draw(ix[j]-1,iy[j]+1,ix[j]-1,iy[j]-1,1);
   draw(ix[j]+1,iy[j]-1,ix[j]-1,iy[j]-1,1);
   draw(ix[j]+1,iy[j]+1,ix[j]-1,iy[j]+1,1)
end;
MPtr:=Fmi;
for i:=1 to ne  do begin
   with MPtr^  do begin
      j:=m1;
      k:=m2;
      draw(ix[j],iy[j],ix[k],iy[k],1)
   end;
   if i <> ne  then MPtr:=MPtr^.nextmi
end;
if nh > 0  then begin
   HPtr:=Fh;
   for i:=1 to nh  do begin
      with HPtr^  do begin
```

```
            j:=T11-1;
            k:=T12;
            MPtr:=Fmi;
            while (j > 0) and (MPtr <> nil)  do begin
               j:=j-1;
               MPtr:=MPtr^.nextmi
            end;
            if k=1
               then l:=MPtr^.m1
               else l:=MPtr^.m2;
            circle(ix[l],iy[l],4,1)
         end;
         MPtr:=MPtr^.next1
      end
   end;
   repeat          until keypressed;
   textmode
end;



{*****************************************************************************
 *                                 OUTFILE                                   *
 *****************************************************************************}
procedure OUTFILE(fn:S10; date:S8; ne,nj,nlc:integer; Fmi,Lmi:MIPtr; Fjc,Ljc,
                  Fh,Lh:T1Ptr; Fxy,Lxy:XYPtr; Fa,La,Fzi,Lzi,Fe,Le,Fcte,
                  Lcte:T2Ptr; Fjl,Ljl,Fpjd,Lpjd:T3Ptr; Fmact,Lmact:MLPtr;
                  Flccv,Llccv:ControlPtr; njc,nh,tnjl,tnmact,tnpjd:integer);
var   i,j              :integer;
      otf              :S14;
      Tmi              :MIPtr;
      Tt1              :T1Ptr;
      Txy              :XYPtr;
      Ta,Tzi,Te,Tcte   :T2Ptr;
      Tjl,Tpjd         :T3Ptr;
      Tmact            :MLPtr;
      Tlccv            :ControlPtr;
{
    Writes the data from the linked lists to the file fn.DAT.
}
begin
   otf:=fn+'.DAT';
   assign(ofn,otf);
   rewrite(ofn);
   writeln(ofn,date);
   writeln(ofn,ne:5,nj:5,nlc:5);
   Tmi:=Fmi;
   for i:=1 to ne do begin
      with Tmi^  do writeln(ofn,m1:5,m2:5);
      Tmi:=Tmi^.nextmi
```

```
end;
writeln(ofn,njc:5);
Tt1:=Fjc;
for i:=1 to njc  do begin
   with Tt1^  do writeln(ofn,T11:5,T12:5);
   Tt1:=Tt1^.next1
end;
writeln(ofn,nh:5);
Tt1:=Fh;
for i:=1 to nh  do begin
   with Tt1^  do writeln(ofn,T11:5,T12:5);
   Tt1:=Tt1^.next1
end;
Txy:=Fxy;
for i:=1 to nj  do begin
   with Txy^  do writeln(ofn,x1:12:4,'  ',x2:12:4);
   Txy:=Txy^.nextx
end;
Ta:=Fa;
Tzi:=Fzi;
Te:=Fe;
Tcte:=Fcte;
for i:=1 to ne  do begin
   writeln(ofn,Ta^.T21:12:4,'  ',Tzi^.T21:12:4,'  ',Te^.T21:12:4,'  ',
         Tcte^.T21:12:4);
   Ta:=Ta^.next2;
   Tzi:=Tzi^.next2;
   Te:=Te^.next2;
   Tcte:=Tcte^.next2
end;
Tlccv:=Flccv;
Tjl:=Fjl;
Tmact:=Fmact;
Tpjd:=Fpjd;
for i:=1 to nlc  do begin
   writeln(ofn,Tlccv^.connjl:5);
   for j:=1 to Tlccv^.connjl  do begin
      with Tjl^  do writeln(ofn,T31:5,T32:5,'  ',T33:12:4);
      Tjl:=Tjl^.next3
   end;
   writeln(ofn,Tlccv^.connmat:5);
   for j:=1 to Tlccv^.connmat  do begin
      with Tmact^  do
         writeln(ofn,mln:5,mlt:5,'  ',mld:12:4,'  ',mld1:12:4,'  ',
               mld2:12:4);
      Tmact:=Tmact^.nextml
   end;
   writeln(ofn,Tlccv^.connpjd:5);
   if Tlccv^.connpjd <> 0  then writeln(ofn,Tlccv^.conpfac:10:1);
   for j:=1 to Tlccv^.connpjd  do begin
```

```
        with Tpjd^  do writeln(ofn,T31:5,T32:5,'  ',T33:12:4);
        Tpjd:=Tpjd^.next3
      end;
      Tlccv:=Tlccv^.nextcon
   end;
   close(ofn)
end;


(*****************************************************************************
 *                                  DELMEM                                   *
 *****************************************************************************)
procedure DELMEM(var Fmi,Lmi:MIPtr; var Fjc,Ljc,Fh,Lh:T1Ptr; var Fxy,Lxy:
                 XYPtr; var Fa,La,Fzi,Lzi,Fe,Le,Fcte,Lcte:T2Ptr; var Fjl,
                 Ljl,Fpjd,Lpjd:T3Ptr; var Fmact,Lmact:MLPtr; var Flccv,
                 Llccv:ControlPtr; ne,nj,nlc,njc,nh,tnjl,tnmact,tnpjd:
                 integer);
var   i              :integer;
      Tmi,Tlmi       :MIPtr;
      Tt1,Tlt1       :T1Ptr;
      Txy,Tlxy       :XYPtr;
      Tt2,Tlt2       :T2Ptr;
      Tt3,Tlt3       :T3Ptr;
      Tml,Tlml       :MLPtr;
      Tlccv,Tllccv   :ControlPtr;
{
    Deletes all the linked lists.
}
begin
   Tmi:=Fmi;
   for i:=1 to ne  do begin
      Tlmi:=Tmi;
      Tmi:=Tmi^.nextmi;
      dispose(Tlmi)
   end;
   Tt1:=Fjc;
   for i:=1 to njc  do begin
      Tlt1:=Tt1;
      Tt1:=Tt1^.next1;
      dispose(Tlt1)
   end;
   Tt1:=Fh;
   for i:=1 to nh  do begin
      Tlt1:=Tt1;
      Tt1:=Tt1^.next1;
      dispose(Tlt1)
   end;
   Txy:=Fxy;
   for i:=1 to nj  do begin
      Tlxy:=Txy;
```

```
   Txy:=Txy^.nextx;
   dispose(Tlxy)
end;
Tt2:=Fa;
for i:=1 to ne  do begin
   Tlt2:=Tt2;
   Tt2:=Tt2^.next2;
   dispose(Tlt2)
end;
Tt2:=Fzi;
for i:=1 to ne  do begin
   Tlt2:=Tt2;
   Tt2:=Tt2^.next2;
   dispose(Tlt2)
end;
Tt2:=Fe;
for i:=1 to ne  do begin
   Tlt2:=Tt2;
   Tt2:=Tt2^.next2;
   dispose(Tlt2)
end;
Tt2:=Fcte;
for i:=1 to ne  do begin
   Tlt2:=Tt2;
   Tt2:=Tt2^.next2;
   dispose(Tlt2)
end;
Tt3:=Fjl;
for i:=1 to tnjl  do begin
   Tlt3:=Tt3;
   Tt3:=Tt3^.next3;
   dispose(Tlt3)
end;
Tml:=Fmact;
for i:=1 to tnmact  do begin
   Tlml:=Tml;
   Tml:=Tml^.nextml;
   dispose(Tlml)
end;
Tt3:=Fpjd;
for i:=1 to tnpjd  do begin
   Tlt3:=Tt3;
   Tt3:=Tt3^.next3;
   dispose(Tlt3)
end;
Tlccv:=Flccv;
for i:=1 to nlc  do begin
   Tllccv:=Tlccv;
   Tlccv:=Tlccv^.nextcon;
   dispose(Tllccv)
```

```
   end;
   Fmi:=nil;     Lmi:=nil;     Fjc:=nil;     Ljc:=nil;
   Fh:=nil;      Lh:=nil;      Fxy:=nil;     Lxy:=nil;
   Fa:=nil;      La:=nil;      Fzi:=nil;     Lzi:=nil;
   Fe:=nil;      Le:=nil;      Fcte:=nil;    Lcte:=nil;
   Fjl:=nil;     Ljl:=nil;     Fmact:=nil;   Lmact:=nil;
   Fpjd:=nil;    Lpjd:=nil;    Flccv:=nil;   Llccv:=nil
end;


{*****************************************************************************
 *                                  ETDOS                                    *
 *****************************************************************************}
procedure ETDOS(fn:S10; date:S8; var Fmi,Lmi:MIPtr; var Fjc,Ljc,Fh,Lh:T1Ptr;
                var Fxy,Lxy:XYPtr; var Fa,La,Lzi,Lzi,Fe,Le,Fcte,Lcte:T2Ptr;
                var Fjl,Ljl,Fpjd,Lpjd:T3Ptr; var Fmact,Lmact:MLPtr;
                var Flccv,Llccv:ControlPtr; j,ne,nj,nlc,njc,nh,nminc,nxy,na,
                nzi,nemod,ncte,nlccv,tnjl,tnmact,tnpjd:integer; var fk:char);
var  i,maxh,maxjt,maxmem:integer;   Tt3:T3Ptr;   Tml:MLPtr;
{
   Calls WRITEFILE.
}
begin
   clrscr;
   if j=1
      then textcolor(14)
      else textcolor(15);
   GTW8(9,1,fn);
   write('.DAT');
   GTW8(64,1,date);
   if j=1
      then GTW9(36,1,'Run FRAME')
      else GTW12(35,1,'Exit to DOS');
   if Lh <> nil
      then maxh:=Lh^.T11
      else maxh:=0;
   Tt3:=Fjl;
   if Fjl <> nil
      then maxjt:=Tt3^.T31
      else maxjt:=0;
   for i:=1 to tnjl  do begin
      if Tt3^.T31 > maxjt  then maxjt:=Tt3^.T31;
      Tt3:=Tt3^.next3
   end;
   Tml:=Fmact;
   if Fmact <> nil
      then maxmem:=Tml^.mln
      else maxmem:=0;
   for i:=1 to tnmact  do begin
      if Tml^.mln > maxmem  then maxmem:=Tml^.mln;
```

```
      Tml:=Tml^.nextml
    end;
    Tt3:=Fpjd;
    for i:=1 to tnpjd  do begin
      if Tt3^.T31 > maxjt  then maxjt:=Tt3^.T31;
      Tt3:=Tt3^.next3
    end;
    if (nminc=ne) and (njc >= 1) and (Ljc^.T11 <= nj) and (maxh <= ne) and
      (nxy=nj) and (na=ne) and (nzi=ne) and (nemod=ne) and (ncte=ne) and
      (nlccv=nlc) and (maxjt <= nj) and (maxmem <= ne)  then begin
      gotoxy(28,9);
      write('Writing file ',fn:10,'.DAT.');
      OUTFILE(fn,date,ne,nj,nlc,Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,
              Fe,Le,Fcte,Lcte,Fjl,Ljl,Fpjd,Lpjd,Fmact,Lmact,Flccv,Llccv,njc,
              nh,tnjl,tnmact,tnpjd);
      GTW36(26,12,'Deleting records from memory.       ');
      DELMEM(Fmi,Lmi,Fjc,Ljc,Fh,Lh,Fxy,Lxy,Fa,La,Fzi,Lzi,Fe,Le,Fcte,Lcte,Fjl,
             Ljl,Fpjd,Lpjd,Fmact,Lmact,Flccv,Llccv,ne,nj,nlc,njc,nh,tnjl,
             tnmact,tnpjd);
      if j=1  then begin
        assign(frame,'FRAME.CHN');
        chain(frame)
      end
    end
    else begin
      gotoxy(19,7);
      write('File ',fn:10,'.DAT cannot be written to disk.');
      gotoxy(13,8);
      write('Data records do not agree with the control variable for:');
      if nminc <> ne  then GTW18(33,9,'MEMBER INCIDENCE  ');
      if (njc < 1) or (Ljc^.T11 > nj)  then GTW18(32,10,'JOINT CONSTRAINTS ');
      if maxh > ne  then GTW6(38,11,'HINGES');
      if nxy <> nj  then GTW18(32,12,'JOINT COORDINATES');
      if (na <> ne) or (nzi <> ne) or (nemod <> ne) or (ncte <> ne)
        then GTW18(32,13,'ELEMENT PROPERTIES');
      if (nlccv <> nlc) or (maxjt > nj) or (maxmem > ne)
        then GTW9(36,14,'LOAD DATA');
      GTW36(23,18,'Hit any key to return to MAIN MENU. ');
      fk:=#59;
      repeat  until keypressed
    end
  end;
```

The vita has been removed
from the scanned document