INVITATION · TO

FORTRAN

FOR · THE

TRS-80

Lawrence L. McNitt

INVITATION TO
FORTRAN
FOR THE
TRS-80

INVITATION TO

FORTRAN

FOR THE

TRS-80

Lawrence L. McNitt

Contents

Preface

FORTRAN was one of the first higher-level languages. Its purpose was to lighten the burden of writing programs for numerical calculations. It is still the primary programming language for scientific computing. FORTRAN was also one of the first languages implemented on more than one computer model, as transportability of programs from one computer model to another is one of its primary virtues.

FORTRAN use spread quickly throughout the late 1950s and early 1960s as many computer manufacturers included FORTRAN on their systems. Advocates of other languages point to the primitive nature of the early FORTRAN implementations while downgrading the usefulness of the language. Periodic revisions to the language result in a language with a remarkable survival rate.

PL/I, BASIC, APL, and PASCAL have all gained acceptance in the scientific community. PL/I and PASCAL were supposed to bury FORTRAN in the archives of ancient history, but FORTRAN lives on. Most existing scientific software uses FORTRAN. New applications continue to use FORTRAN.

FORTRAN is widely available. There is a high degree of standardization for the versions of the language implemented by the computer manufacturers. The scientific community is familiar with the language. Most scientific organizations have an existing, comprehensive library of FORTRAN subroutines.

FORTRAN is available for most microcomputers that have disk drives. These versions are quite complete with the exception of complex number representation. They include the microcomputer, PEEK and POKE capability, and a choice of eight-bit or 16-bit binary integers. They provide standard single and double precision floating point numbers.

This book introduces the concepts and practices of FORTRAN programming in the context of Microsoft FORTRAN for

the RADIO SHACK TRS-80. This book does not duplicate the
FORTRAN manual that comes with the FORTRAN software.
It is a learning tool that presents the material in a step-by-step
fashion. Included are numerous examples illustrating features
of the language. The example programs demonstrate program-
ming methods and style. This book is a valuable aid for self-study,
and is useful as a supplementary text in computer science, en-
gineering, mathematics, and applied science disciplines.

The examples demonstrate the usefulness of FORTRAN for
realistic problem situations. FORTRAN is at its best when used
with a comprehensive scientific subroutine library. Included are
discussions relative to the development and use of such libraries.
The discussions do not presume an extensive background in
mathematics. On the other hand, the book does not ignore im-
portant mathematical applications such as solving simultaneous
linear equations.

1 Introduction

OVERVIEW

FORTRAN is an algebraic language for numerical computing. Simple computations involve addition, subtraction, multiplication, division, and exponentiation. Complex calculations involve long sequences of these simple calculations.

Using FORTRAN on any computer requires some experience with the computer system. Steps in the programming process include creating the source program, compiling it, and running the program.

The program should produce readable output that is easily understood. The source program must be readable by both the machine and the programmer. The machine compiles the source program to obtain the object program. Other programmers may be called to modify it at some later date.

☐

1.1 Primitive Output

SIMPLE ARITHMETIC

Simple computations include addition, subtraction, multiplication, division, and exponentiation (powers and roots). The following table lists the FORTRAN symbols for these operations:

+   Addition

−   Subtraction

*   Multiplication

/   Division

** Exponentiation

1

The expression

$$2.75 * 14.3$$

computes the product of 2.75 and 14.3. The expression

$$.988 ** 16$$

raises .988 to the 16th power.

PRECEDENCE ORDERING

FORTRAN follows the traditional precedence ordering of arithmetic operations. Exponentiation takes precedence over multiplication and division which take precedence over addition and subtraction. The expression

$$3 * 4 - 27 / 3$$

subtracts the ratio 27/3 from the product 3 * 4.

PARENTHESES

Parentheses override the precedence ordering. The expression

$$3 * (8 - 4) / 6$$

performs the operation within the parentheses first. Operations on a given precedence level proceed from left to right.

UNARY MINUS

The unary minus negates a value or an expression. The expression

$$3 * 4 / - (1 + 5)$$

negates the result in the parentheses. Some FORTRAN compilers do not permit two operation symbols to be adjacent. The outer parentheses in the expression

$$3 * 4 / (- (1 + 5))$$

are necessary for those systems.

VARIABLES

FORTRAN variables contain values for use in the program. These may be initial values (parameters) describing the problem situa-

tion or they may be computed values (solution) for the problem. They may be intermediate computed values needed temporarily during the solution process.

FORTRAN variable names consist of from one to six symbols. The first symbol must be a letter, and the following symbols can be letters or digits. The name should reflect the intended use of the variable, but any name is possible.

INTEGER VERSUS REAL

FORTRAN distinguishes between integer and real values. The expression

$$12.89 * 43.5$$

computes the product of two real values. The expression

$$12 * 43$$

computes the product of two integers. The expression

$$12.89 * 43$$

is a mixed-mode expression.

FORTRAN programs should not contain mixed-mode expressions except for exponentiation. The expression

$$12.75 ** 4$$

is acceptable. Not all FORTRAN compilers translate mixed-mode expressions into efficient machine instructions.

VARIABLE NAMES

FORTRAN variable names beginning with the letters *I* through *N* are integers. Names beginning with other letters are real. The names

APPLE   GRAPES   X123   TOTAL

designate real values. The names

NUMBER   INDEX   KTOT   L12

designate integers.

TYPE SPECIFICATION

FORTRAN allows variable type specification. The specification command comes at the beginning of the program. The command

INTEGER APPLE, GRAPES, TOTAL

overrides the naming convention for real variables. The command

REAL NUMBER, INDEX, L12

overrides the naming convention for integer variables.

     Readability is extremely important. The distinction between integer and real variables is significant. If type specification is used, then all variables should be typed and this practice followed for all programs. Overriding the naming conventions for just two or three variables in a large program will result in a program that is extremely difficult to debug and to modify later.

PROCESSING STEPS

Three steps are necessary in preparing a FORTRAN program for execution. The first step involves the creation and editing of the FORTRAN source program. The second step uses the FORTRAN compiler to create a relocatable object program. The third step uses the linkage editor to convert the relocatable object modules to executable command module form for immediate execution or for later execution.

EDITING

The Microsoft FORTRAN package for the Radio Shack TRS-80 computers includes an EDIT program for creating and editing the source programs. More sophisticated editors such as SCRIPSIT are much easier to use for this purpose. SCRIPSIT requires the S,A option giving the ASCII storage mode. The SCRIPSIT command

S,A PROG/FOR

saves the file in the workspace as file PROG/FOR in ASCII form acceptable to the FORTRAN compiler.

FORTRAN programs follow a strict layout. Columns 1 through 6 are reserved for statement numbers, comment symbols, and continuation symbols. The letter C in column 1 designates a comment line for documenting the source program. Any symbol in column 6 designates a continuation of the previous line. FORTRAN program statements reference other lines through statement numbers. The statement numbers may be located within columns 1 through 5.

FORTRAN program statements begin on or after column 7. Many FORTRAN programmers begin the command portion of the line in column 8 or 10. This aids readability and reduces the potential confusion with the continuation symbol in column 6.

FORTRAN source programs should have the file extension FOR. The file name

PROG/FOR

is typical.

COMPILATION

The FORTRAN compiler reads the source program and creates the corresponding object program. The Microsoft FORTRAN compiler has the command file name F80. The command

F80 PROG=PROG

is typical. This calls the compiler, compiles the source program in PROG/FOR, and places the relocatable object program in PROG/REL. The compiler expects the standard file name extensions.

The Microsoft compiler will create a listing file for those familiar with assembly and machine languages. The command

F80 PROG,PROG=PROG

reads the source program from PROG/FOR, places the relocatable object program into the file PROG/REL, and places the FORTRAN listing into the file PROG/LST.

LINKAGE EDITOR

The linkage editor (Linker) creates an executable program from the relocatable object modules. L80 is the name of the TRSDOS command file containing the linker. Both the compiler and the linkage editor contain switches designating options. Three switches are necessary for the linker:

G Execute the program

N Save program as command file

E Exit linker to TRSDOS

The command

L80 PROG-G

reads the relocatable file PROG/REL, forms the executable program, and executes it.

The usual practice is to save the executable program as a command file. The command

L80 PROG-N,PROG-E

reads the relocatable file PROG/REL, places the executable program into the command file PROG/CMD, and then exits the linker. While in TRSDOS the command

PROG

loads and executes the command file PROG/CMD.

COMMENTS

Compiled languages require extra steps compared to interpreters such as BASIC. Most systems require the two-stage compilation involving the initial compiler and the following linker. Linking

subroutines into the main program requires the linking step. Large programs may contain several separately compiled subroutines. The linker combines the main program relocatable with the sub-routine relocatables to form the complete, executable program load module. The command

L80 PROG-N,SUB1,SUB2,PROG-E

combines the relocatables PROG/REL, SUB1/REL, and SUB2/REL to form the executable load module PROG/CMD.

The compilation and linkage stages are time-consuming for large programs. The steps of editing, compiling, and linking become tedious during the debugging process. The resulting object program, however, executes much more quickly than interpreted programs. This is its greatest asset.

The original documented source program is not in memory at run time. Commerical software packages may contain only the executable command files. Software piracy is a difficult problem to address with microcomputers. A secondary benefit of compiled programs is the separation of source programs from object programs. Competitors will not have easy access to the source programs to adapt for their own products.

READING AND WRITING

Input and output consist of reading and writing files. The statement

READ(9,124) APPLES, NUMBER

is a typical read statement. The names APPLES and NUMBER are variable names. The read operation places values in those variables.

The read statement contains one or more parameters. The first parameter is a unit number. The system relates the unit number to a particular device or external file. The Microsoft FORTRAN for the Radio Shack TRS-80 uses the following convention for unit numbers:

1,3,4,5 Terminal screen or keyboard

2   Line printer

6	Disk file FORT06/DAT
7	Disk file FORT07/DAT
8	Disk file FORT08/DAT
9	Disk file FORT09/DAT
10	Disk file FORT10/DAT.

There are other ways of linking disk data files in addition to these.

The second parameter within the parentheses designates an optional format statement number. The format statement specifies the exact format of the data. If omitted, the data is in an internal unformatted form. All numerical data sent to the printer must be formatted. Other optional parameters will be discussed later.

SIMPLE FORMATTING

Microsoft FORTRAN uses standard format statements. The most common numeric formats use the *I* symbol for integer and the *F* format for real values. The expression

I10

defines an integer field of ten positions. The expression

F10.2

defines a field of 10 positions for a real number. The decimal point will be two places from the right.

Each expression may be repeated. The expression

5I10

specifies five fields of 10 columns each for integer data. The expression

3F15.5

specifies three fields of 15 columns each with five digits to the right of the decimal point.

The specification fields are in any order and size. The expression

I5,F12.2,3I10

defines an integer field of five positions, a field of 12 positions for a real value, and three fields of 10 positions each for integer data.

The symbol X specifies empty space and may be repeated. The expression

I5,5X,F10.2

separates the integer field and the real field with five blank positions.

The FORMAT statement includes the statement number given as the first parameter of the READ or WRITE statement. The statement

124 FORMAT (F10.2,5X,I5)

is typical.

AREA AND CIRCUMFERENCE OF A RECTANGLE

The area of a rectangle is the product of the length and the width. The circumference is twice the sum of the length and width. Let L be the length and W the width. The expression

A = L * W

gives the area A. The expression

C = 2 * (L + W)

gives the circumference C.

FORTRAN VARIABLE NAMES

Good practice requires selecting names that are meaningful. Variable names should follow the FORTRAN naming conventions for variable types. If the data values or their intermediate results include fractions, then the variables and constants should be real.

The name AREA is natural for the area of the rectangle. The name CIRCUM is possible for the circumference. FORTRAN limits the size of the name to a maximum of six symbols.

COMPUTATION

A rectangular garden plot has a length of 40 feet and a width of 30 feet. The statement

AREA = 40.0 * 30.0

computes the product and places it in the variable AREA. The statement

CIRCUM = 2.0 * (40.0 + 30.0)

places the value of the circumference into the variable CIRCUM.

OUTPUT

The unit number 2 designates the line printer. The two output variables are AREA and CIRCUM. The two statements

```
    WRITE(2,10) AREA,CIRCUM
10  FORMAT(2F10.2)
```

print the results on the line printer. The output will consist of two fields of 10 columns each with two values to the right of the decimal point.

STOP AND END

The last line of the FORTRAN program consists of the END statement. The last executable statement is the STOP statement. These statements should be included in each program.

FORTRAN SOURCE PROGRAM

The following is the complete FORTRAN source program computing the area and circumference of a rectangle having length 40.0 and width 30.0:

```
   AREA = 40.0 * 30.0
   CIRCUM = 2.0 * (40.0 + 30.0)
   WRITE(2,10) AREA,CIRCUM
10 FORMAT(2F10.2)
   STOP
   END
```

OUTPUT

The resulting output is sent to the line printer:

1200.00   140.00

This primitive program illustrates the rudiments of FORTRAN programming. The assignment statements perform calculations and place the results into the variable specified to the left of the equals symbol. The output statements include the WRITE command and the FORMAT statement.

1.2   Number Representations

PRINTING SYMBOLS

Most computers use eight-bit bytes to represent basic symbols for input, output, and communication. These symbols include upper and lower case letters, numeric digits, special symbols, and control characters.

INTERNAL NUMBERS

Internal numeric quantities do not generally use the printable numeric digits. Internal representations include signed binary integers and real numbers which include single and double precision. Microsoft FORTRAN also includes three sizes of signed binary integers.

BINARY INTEGERS

The standard binary integer for Microsoft FORTRAN is the 16-bit integer. The value must fall between –32768 and 32767. The standard integer contains two bytes. An extended integer contains four bytes or 32 bits. The specification statement

INTEGER*4 NUMBER, ITOTAL

designates the variables NUMBER and ITOTAL as extended integers.

Another representation specifies an integer of byte length (eight-bits). The value must fall between –128 and 127. Bytes take much less room in storage than standard or extended integers. The value must fall within a limited range. The specification statement

BYTE NUMBER, KETTLE

defines the variables NUMBER and KETTLE to be byte length.

TWO'S COMPLEMENT

Microsoft FORTRAN uses two's complement binary integers. The following table gives the representations for byte-length integers near zero:

Number	Byte-length binary number
-3	11111101
-2	11111110
-1	11111111
0	00000000
1	00000001
2	00000010
3	00000011

REAL NUMBERS

Real numbers are in an internal scientific notation form. Each number contains a mantissa giving the fractional part of the

number and an exponent. Single precision real values require four bytes of storage. They provide the equivalent of seven significant decimal digits of precision.

Double-precision real numbers require eight bytes of storage and provide the equivalent of 16 digits of precision. In either case the value must fall within the range 10**–38 and 10**38 in magnitude. The specification statement

DOUBLE PRECISION APPLES, VALUE

assigns the type specification double precision to the variables APPLES and VALUE.

NUMERIC LITERALS

Literals appear as numbers within the program listing. The values

$$123 \quad -12425 \quad 13000 \quad -14$$

are literal integers. The values

$$12.75 \quad 3.6 \quad 19.2356$$

are real literals.

Scientific notation is also possible for numeric literals. A symbol represents the start of the exponent portion of the number. The letter E specifies single precision and the letter D specifies double precision. The value

$$3.45E6$$

represents the value 3.45 times 10 to the sixth power in single precision form. The value

$$1.23145D-5$$

represents the value 1.23145 times 10 to the –5th power in double precision form.

OUTPUT TO VIDEO SCREEN

The typical microcomputer system using the FORTRAN language will include two disk drives, a printer, and a terminal facility containing video screen and keyboard. Any output to be saved

should be printed. The user, however, sits at the terminal typing in commands and watching the screen. Programs running in this manner may send output to the screen rather than to the printer.

In Microsoft FORTRAN for the TRS-80 unit, number 2 designates the printer and unit number 1 designates the terminal. The program of this section sends the output to the terminal rather than the printer.

PROGRAM

The following program sends the output to the terminal and uses integer variables:

```
      LENGTH = 40
      LWIDTH = 30
      LAREA = LENGTH * LWIDTH
      LCIR = 2 * (LENGTH + LWIDTH)
      WRITE(1,200) LAREA, LCIR
200   FORMAT(2I10)
      STOP
      END
```

TEST RUN

The following shows the video output generated during the test run:

```
P0102
      1200          140
STOP
```

DUAL COMMAND

Radio Shack TRSDOS contains the DUAL command which duplicates the video screen output with the printer to obtain a permanent copy. This is a useful command. It reduces the need for two versions of the program—one for the video screen and the other for the printer.

1.3   Messages

LABEL OUTPUT

All output to the terminal or to the printer must be labeled. This reduces the risk of misinterpreting the output. The programs of the first two sections compute the area and circumference of a rectangle. Somebody unfamiliar with the program could easily confuse the two values.

Label messages for output values consist of short names or explanations. The labels "AREA" and "CIRCUMFERENCE" are sufficient. Labels and messages form another data type for FOR-TRAN. They may be called alphabetic data or character strings. FORTRAN string literals are enclosed in single quotes. The strings

'HELLO'   'THE ANSWER IS'

are typical.

FORMAT STATEMENTS

String literals are part of the FORMAT statement. The statement

10 FORMAT(' AREA',F10.2)

places the identifying label AREA in the field defined by the first five positions. The following ten positions will contain the value of the real variable specified in the corresponding WRITE command.

An alternate form

10 FORMAT(5H AREA,F10.2)

accomplishes exactly the same thing. The H indicates Hollerith data. This form clearly specifies the length of the string, but it requires more work initially. For complex FORMAT statements having many entries, it may save time during the debugging of the program.

CARRIAGE CONTROL CHARACTERS

The first character sent to the printer is a carriage control character. A space results in printing on the next line. A value of zero results in skipping one line before printing. The value 1 advances the printer to the top of the next page. These are the universal symbols employed by FORTRAN for most computer systems.

These characters may not work for sending output to the video screen. The space and zero options will work for the Radio Shack FORTRAN. The option advancing to the next page will not work with the video terminal.

FORTRAN FORMAT statements include another method for advancing printer lines. The slash, "/", begins the next line. A series of slashes together, "///", advances the stated number of lines.

The FORMAT statement

10 FORMAT(' AREA,F10.2)

prints the result on the next line. The statement

10 FORMAT('OAREA',F10.2)

skips a line before printing. The statement

10 FORMAT(/' AREA',F10.2)

also skips a line before printing. The statement

10 FORMAT(///' AREA',F10.2)

skips three lines before printing.

CONTINUATION LINES

Any character in column six signifies a continuation of the previous line. The primary use is for continuing FORMAT statements. Common practice is to include a numeric digit as the continuation symbol, but any symbol will work.

The following FORMAT statement prints the area and circumference with labels on two lines:

```
10 FORMAT(' AREA               ',F10.2
   2         /' CIRCUMFERENCE   ',F10.2)
```

The WRITE command

```
WRITE(1,10) AREA, CIRCUM
```

references the FORMAT statement.

IDENTIFYING MESSAGE

Printed output should also contain identifying messages. These may include the name of the program and a statement of its purpose. If the output is to be filed away for reference, these messages are vital.

The program name should follow the conventions of the organization for its program libraries. The program library may contain hundreds of programs. An index to those libraries will list the programs by names and include a short statement of purpose for each. This aids in locating the needed program.

Problems will arise. At times it may appear that the program has finished when it has not. A short message signifying the normal end of the program gives reassurance to the user.

DESTINATION OF MESSAGES

Typically, the program name, statement of purpose, and final message go to the video screen. Sometimes they are sent to the printer, and sometimes they may be routed to both the screen and to the printer.

PROGRAM

The following program includes the initial and final messages and sends them to the printer.

```
    WRITE(2,10)
10  FORMAT(' PROGRAM P0103')
```

```
      WRITE(2,20)
20    FORMAT(' COMPUTE THE AREA'
  2            /' AND CIRCUMFERENCE'
  3            /' FOR A RECTANGLE.')
      ALNGTH = 40.0
      WIDTH = 30.0
      AREA = ALNGTH * WIDTH
      CIRCUM = 2.0 * (ALNGTH + WIDTH)
      WRITE(2,30) AREA, CIRCUM
30    FORMAT(' AREA            ',F10.2
  2            /' CIRCUMFERENCE  ',F10.2)
      WRITE(2,40)
40    FORMAT(' END OF PROGRAM')
      STOP
      END
```

PRINTED OUTPUT

The following is the printed output produced by the program:

```
PROGRAM P0103
COMPUTE THE AREA
AND CIRCUMFERENCE
FOR A RECTANGLE.
AREA                1200.00
CIRCUMFERENCE        140.00
END OF PROGRAM
```

1.4   White Space

READABILITY

Output that is too dense or jumbled together is hard to interpret. Readability is an important goal. Output will naturally fall into distinct sections. Blank lines between sections draw attention to these natural divisions and enhance the readability of the output. The resulting white space gives a neat, pleasing appearance to the output. Organized output is easier to interpret.

PAGING

Another technique is to advance to the top of the next page for major divisions in the output. Using the digit 1 as the carriage

control character accomplishes the page eject on the printer. This technique should not be used excessively. The output could become distributed over too many pages with few items per page.

PROGRAM

The following program includes blank lines for white space:

```
     WRITE(1,10)
10   FORMAT(' PROGRAM P0104')
     WRITE(2,20)
20   FORMAT('1PROGRAM P0104'
  2             /'0COMPUTE THE AREA'
  3             /' AND CIRCUMFERENCE'
  4             /' FOR A RECTANGLE.')
     ALNGTH = 40.0
     WIDTH = 30.0
     AREA = ALNGTH * WIDTH
     CIRCUM = 2.0 * (ALNGTH + WIDTH)
     WRITE(2,30) AREA, CIRCUM
30   FORMAT('0AREA                  ',F10.2
  2             /' CIRCUMFERENCE    ',F10.2
  3             /'0END OF OUTPUT       '
  4             /'1')
     WRITE(1,40)
40   FORMAT(' END OF PROGRAM'/)
     STOP
     END
```

The carriage control character '0' skips a line before printing.

TEST RUN VIDEO OUTPUT

The following gives the video output of the test run:

PROGRAM P0104
END OF PROGRAM

TEST RUN PRINTED OUTPUT

The following gives the printed output of the test run:

PROGRAM P0104

```
COMPUTE THE AREA
AND CIRCUMFERENCE
FOR A RECTANGLE.
AREA                  1200.00
CIRCUMFERENCE          140.00
END OF OUTPUT
```

1.5   Documentation

READABLE PROGRAMS

FORTRAN source programs must be readable by both the computer and the programmer. Readability is just as important for source programs as it is for printed output. The program naturally divides into sections just as the printed output does.

SECTION IDENTIFICATION

White space works well with printed output. Another technique calls attention to sections within a source program listing. This technique consists of naming each section of the source program and placing a box of asterisks around each section name. The goal is to make the program easy to read. This makes the program organization highly visible.

COMMENT LINES

FORTRAN comment lines start with the letter C in column 1. The comment lines

```
C   COMPUTE THE AREA
C   AND CIRCUMFERENCE
C   FOR A RECTANGLE
```

are typical.
      The following is a typical section name surrounded by a box of asterisks:

```
C   ***********************************
C   *     INITIAL MESSAGE             *
C   ***********************************
```

Readability is enhanced by drawing attention to the sections of the program.

VARIABLE DICTIONARY

FORTRAN limits variable names to a maximum of six characters. This limits the meaning of the names. A variable dictionary is highly recommended. This dictionary gives an expanded explanation for each variable in the program. Although many programmers omit this documentation, it does help those programmers who may need to modify the program later.

STATEMENT OF PURPOSE

A short statement of purpose is helpful. This is placed as a comment together with the name of the programmer and other identifying information at the beginning of the source program.

COMPUTER SYSTEM

If all the computers were entirely consistent in their use of FORTRAN, it would be convenient, but each system has its quirks. A short statement giving the computer system for which the source program was written and tested is desirable. Some changes may be necessary before the program will run on another machine.

DOCUMENTATION

Documentation is essential for any program that will be used by various people at different times. The most important part of any documentation package is a listing of the source program. This source program should contain extensive comments giving the statement of purpose, variable dictionary, and section names.

The documentation will also include test runs giving results for known problems. These test runs form an important part of any conversion effort. After modifying the program or converting the program to run on another computer, the test runs validate the converted program.

The documentation package may also include a user's guide giving instructions to new users. Complex systems may include an operator's manual if the program is normally executed on a central computer by the regular staff of computer operators.

PROGRAM

The following program includes extensive comments identifying
program sections and giving information about the program:

```
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *      P0105                                              *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    AUTHOR
C        COPYRIGHT 1982
C        BY LAWRENCE MCNITT.
C    PURPOSE
C        AREA AND CIRCUMFERENCE
C        FOR A RECTANGLE.
C    SYSTEM
C        MICROSOFT FORTRAN
C        RADIO SHACK TRS-80.
C        MODEL III.
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *      ORGANIZATION                                       *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    INITIAL MESSAGE
C    PROCESS
C    OUTPUT
C    FINAL MESSAGE
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *      VARIABLES                                          *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    ALNGTH        LENGTH OF RECTANGLE
C    WIDTH         WIDTH OF RECTANGLE
C    AREA          AREA
C    CIRCUM        CIRCUMFERENCE
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *      INITIAL MESSAGE                                    *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         WRITE(1,10)
   10    FORMAT(' PROGRAM P0105')
         WRITE(2,20)
   20    FORMAT('1PROGRAM P0105'
    2             /'0COMPUTE THE AREA'
    3             /' AND CIRCUMFERENCE'
    4             /' FOR A RECTANGLE.')
```

```
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     *     PROCESS                                           *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
            ALNGTH = 40.0
            WIDTH = 30.0
            AREA = ALNGTH * WIDTH
            CIRCUM = 2.0 * (ALNGTH + WIDTH)
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     *     OUTPUT                                            *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
            WRITE(2,30) AREA, CIRCUM
      30    FORMAT('0AREA                    ',F10.2
       2              /' CIRCUMFERENCE    'F10.2)
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     *     FINAL MESSAGE                             *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
            WRITE(2,40)
      40    FORMAT('0END OF OUTPUT     '
       2              /'1')
            WRITE(1,50)
      50    FORMAT(' END OF PROGRAM'/)
            STOP
            END
```

Documentation is essential for production programs that are to be used repetitively.

OUTPUT FOR VIDEO TERMINAL

The following gives the output to the video terminal:

PROGRAM P0105
END OF PROGRAM

In this case the video output provides a record of the running of the program.

PRINTED OUTPUT

The following gives the printed output:

PROGRAM P0105

```
COMPUTE THE AREA
AND CIRCUMFERENCE
FOR A RECTANGLE.
AREA                  1200.00
CIRCUMFERENCE          140.00
END OF OUTPUT
```

The printed output includes blank lines for readability.

1.6   Exercises

1. Write a documented program computing the velocity in feet per second and distance in feet of an object accelerating at the constant rate of eight feet/sec/sec for 20 seconds. The formula

$$V = A * T$$

gives the velocity in feet per second and

$$D = .5 * A * T ** 2$$

gives the distance in feet.

2. Write a documented program computing the future value of $1,275.25 at the end of five years in an account earning interest at the rate of 7.75 percent compounded quarterly. The formula

$$F = P * (1 + R / Q) ** (Q * N)$$

gives the future value assuming compouding Q times per year and an interest rate of R.

3. A cylindrical water tank is 10 feet long and six feet in diameter. Write a documented program computing the contents in gallons and the net weight in pounds. The formula

$$A = 3.14159 * R * R$$

gives the area of the end of the tank as a function of the radius. Multiplying this by the length gives the volume in cubic feet. Multiplying by the number of pounds per cubic foot of water gives the net weight in pounds.

4. The cost is $24 per foot of fence for a garden plot. The dollar yield for the plot will be $4.75 per square foot. Determine the total cost of fencing and the total dollar yield for a garden plot that is 40 feet long and 30 feet wide.

5. A cylindrical tank is 10 feet long and six feet in diameter. Paint for the exterior costs $8.75 per gallon. How many gallons are needed and how much will the cost be?

6. The mathematical constant *e* is about 2.7183. The term

$$(1 + 1 / N) ** N$$

approaches *e* as *N* increases. Estimate the value of the constant *e* using an *N* of 1000.

7. Numerically estimate the slope of the function

$$Y = 3.5 * X ** 2 - 4 * X + 25$$

at the point *X* = 2. Evaluate the function at the points *X* = 2 and *X* = 2 + *D*, giving *Y*1 and *Y*2, respectively. The term

$$(Y2 - Y1) / D$$

is approximately equal to the slope of the function at the point in question.

2 Iteration

OVERVIEW Iteration is a powerful tool for computer programming languages. All programming languages provide statements for looping and repeating a section of the program. The FORTRAN Do-loop provides iteration capability.

This chapter explores looping in the context of mathematical sequences and series. Again, the emphasis is on writing readable programs. Even at best, iterative procedures are difficult to understand. Every effort should be expended to improve readability.

□

2.1   Do-loop

REPETITION

All programming languages provide the capability of repeating a section of the program. In FORTRAN the Do-loop performs this task. The DO statement defines the beginning of the loop. It contains a line number defining the end of the loop, an index variable used for the looping process, and loop parameters controlling the values assigned to the variable.

The statement

DO 112 I = 1, 10

is typical. The scope of the loop includes all executable statements through the statement numbered 112. Variable I is the index variable. The value 1 is the first value, and 10 is the last value. The increment is assumed to be 1. The loop is repeated

27

10 times with the variable *I* containing the value 1 the first time, the value 2 the second time, etc.

The index variable and the loop parameters must be integers. An optional increment can be included. The statement

DO 174 IVALUE = 10,80,5

sets IVALUE to the values 10, 15, 20, . . ., 80 in turn. The loop parameters must be positive integers.

END OF THE LOOP

The statement number given in the DO command identifies the end of the loop. The last line must be an executable statement other than a branch or STOP. A CONTINUE statement is often used for this purpose.

TABLE GENERATION

The first computer applications during World War II and immediately following consisted of generating mathematical tables. This is still an important application. The table consists of columns of numbers with column and row headings which aid in interpreting the table.

TABLE OF SQUARES

The simplest example of table generation consists of generating a table of squares for the first few integers. The DO command

DO 20 IVALUE = 1, 10

performs the loop ten times, assigning the values 1, 2, . . ., 10 to the index variable IVALUE. Statement number 20 is the last line of the loop.

The variable IVALUE is available for use by the program but it may not be changed by statements within the loop. Failure to follow this requirement will result in programs that will not work reliably on different versions of the FORTRAN compiler.

The following program section generates the table of squares for the first ten integers:

```
    DO 20 IVALUE = 1, 10
    ISQUAR = IVALUE ** 2
    WRITE(2,10) IVALUE, ISQUAR
10  FORMAT(2I10)
20  CONTINUE
```

The primary use of CONTINUE statements is as the last line of the loop.

SUMMATION

Summing a set of values is a common task. One variable is chosen to be the accumulator. The first task is to clear the accumulator to zero. Then the values are summed by adding them to the accumulator. If ISUM is an integer accumulator and IVALUE is the current value, then the command

```
ISUM = ISUM + IVALUE
```

adds the contents of IVALUE to the variable ISUM. This expression does not represent an algebraic identity. Within programming languages, this indicates that the result should be placed in the variable to the left of the equals sign.

The following loop illustrates the summing operation by computing the sum of the squares of the first 10 integers:

```
    ISUM = 0
    DO 10 IVALUE = 1, 10
10  ISUM = ISUM + IVALUE ** 2
```

This loop illustrates using a statement other than a CONTINUE as the last line of the loop.

COUNTER

The loop provides a mechanism for counting the number of repetitions of the loop. The statement

```
DO 24 I = 1, 100
```

performs the loop one hundred times. If the variable I is not referenced inside the loop, then it serves as a counter. This is used if a section of the program must be repeated a specified number of times.

VARIABLES AS PARAMETERS

The loop parameters may be integer variables. The command

DO 124 IVALUE = ISTART, ISTOP, ISTEP

uses variables for all three parameters. The command

DO 640 ICOUNT = 1, NUMBER

performs the loop the number of times given in the variable NUMBER.

Microsoft FORTAN for the Radio Shack TRS-80 does not allow extended integers (32-bit) for the index variable of the Do-loop.

NESTED LOOPS

FORTRAN allows nested loops. The inner loop must be entirely nested within the outer loops. Both inner and outer loops may have the same ending statement although this reduces readability.

FUTURE VALUE OF AN ANNUITY

Pension plans may consist of regular payments into an annuity. The future accumulated value of the annuity is an important consideration for those preparing for retirement. The deposits are made regularly and earn compound interest.

Assume that monthly deposits of $50.00 are made into an account earning 12.75 percent compounded monthly. What is the future value of this account? Listing the future value at the end of each deposit would give too much detail. Listing the accumulated value at the end of each year is reasonable.

Formulas exist for calculating the future value of an annuity without tracing the individual deposits. To illustrate looping, the

programs of this chapter calculate the interest with each monthly deposit. This requires a nested Do-loop. The inner loop accumulates each of the twelve monthly deposits during the year. The outer loop processes each year.

The formula

$$A = B * (1 + .01 * R / 12)$$

gives the amount of interest for the month assuming a beginning account balance of **B** and an annual percentage interest of **R** with monthly compounding.

PROGRAM

The following program generates a year-end summary for an annuity for each of the first ten years:

```
C     *************************************
C     *    P0201                           *
C     *************************************
C     AUTHOR
C          COPYRIGHT 1982
C          BY LAWRENCE MCNITT.
C     PURPOSE
C          FUTURE VALUE AT END
C          OF EACH YEAR FOR
C          REGULAR DEPOSITS.
C     SYSTEM
C          MICROSOFT FORTRAN
C          RADIO SHACK TRS-80
C          MODEL III.
C     *************************************
C     *    ORGANIZATION                    *
C     *************************************
C     INITIAL MESSAGE
C     PROBLEM PARAMETERS
C     HEADING
C     PROCESS
C     FINAL MESSAGE
C     *************************************
C     *    VARIABLES                       *
C     *************************************
```

```
C    NYEARS    NUMBER OF YEARS
C    NPER      NUMBER OF PERIODS PER YEAR
C    IYEAR     CURRENT YEAR
C    IPER      CURRENT PERIOD
C    AMOUNT    AMOUNT OF REGULAR DEPOSIT
C    BAL       CURRENT BALANCE
C    YRATE     YEARLY RATE  (PERCENT)
C    PRATE     PERIOD RATE  (FRACTION)
C    ************************************
C    *    INITIAL MESSAGE                *
C    ************************************
         WRITE(2,100)
  100    FORMAT('1PROGRAM'
     2             //' COMPUTE THE FUTURE VALUE'
     3             /' AT THE END OF EACH YEAR'
     4             /' FOR REGULAR DEPOSITS.')
C    ************************************
C    *    PROBLEM PARAMETERS             *
C    ************************************
         NYEARS = 10
         NPER   = 12
         AMOUNT = 50.0
         YRATE  = 12.75
         PRATE  = .01 * YRATE / NPER
C    ************************************
C    *    HEADING                        *
C    ************************************
         WRITE(2,200) AMOUNT, NPER, YRATE
  200    FORMAT(/' AMOUNT OF DEPOSIT PER PERIOD ',F10.2
     2             /' NUMBER OF PERIODS PER YEAR   ',I7
     3             /' ANNUAL INTEREST RATE         ',F13.5
     4             //' YEAR     BALANCE')
C    ************************************
C    *    PROCESS                        *
C    ************************************
         BAL = 0.0
         DO 390 IYEAR = 1, NYEARS
         DO 370 IPER = 1, NPER
  370    BAL = BAL * (1.0 + PRATE) + AMOUNT
         WRITE(2,380) IYEAR,BAL
  380    FORMAT(I5,F10.2)
  390    CONTINUE
```

```
C     *****************************************
C     *     FINAL MESSAGE                      *
C     *****************************************
      WRITE(2,410)
410   FORMAT(///' END OF OUTPUT')
      STOP
      END
```

PRINTED OUTPUT FROM TEST RUN

The following is the printed output produced by the test run:

PROGRAM

COMPUTE THE FUTURE VALUE
AT THE END OF EACH YEAR
FOR REGULAR DEPOSITS.

AMOUNT OF DEPOSIT PER PERIOD	50.00
NUMBER OF PERIODS PER YEAR	12
ANNUAL INTEREST RATE	12.75000

YEAR	BALANCE
1	636.33
2	1358.71
3	2178.78
4	3109.73
5	4166.56
6	6366.31
7	6728.28
8	8274.42
9	10029.63
10	12022.18

END OF OUTPUT

READABILITY AND STYLE

Iterative procedures can be difficult to understand. Every effort is needed to make iterative programs readable. Both the programmer writing the initial program and later programmers modifying the program should be familiar with the subject area. Style is as important in writing programs as it is in writing articles for pub-

lication. Some features of style including documentation within the listing have already been discussed. The next section introduces indentation to aid in representing loops.

2.2 Indentation

READABILITY

Readability is crucial for FORTRAN source programs. One technique that aids readability is to indent the inner part of the loop several spaces. This is easily done when writing the program. It makes the scope of the loop much more visible.

NOT INDENTED

The following program generates a table of squares for the first ten integers:

```
      DO 20 IVALUE = 1, 10
      ISQUAR = IVALUE ** 2
      WRITE(2,10) IVALUE, ISQUAR
10    FORMAT(2I10)
20    CONTINUE
      STOP
      END
```

The inner part of the loop is not indented.

INDENTED

The following program also generates the table of squares for the first ten integers:

```
      DO 20  IVALUE = 1, 10
          ISQUAR = IVALUE ** 2
          WRITE(2,10) IVALUE, ISQUAR
10          FORMAT(2I10)
20    CONTINUE
      STOP
      END
```

The inner statements of the loop are indented several columns, drawing attention to the statements that are repeated in the loop.

COMMENTS

Indenting is another technique for readability. The program will not run any better. The purpose is to make the program more readable for the maintenance programmers. Indenting helps the person reading the program to find the end of the loop. Without this visual tool the programmer must search for the statement number defining the end of the loop. Consistent use of indenting requires a CONTINUE statement defining the end of the loop. This CONTINUE statement should be on the same level as the corresponding DO statement.

PROGRAM

The following program computes the future values of an annuity at the end of each of the first few years using loop indenting and commenting for readability:

```
C    ************************************
C    *    P0202                         *
C    ************************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         FUTURE VALUE AT END
C         OF EACH YEAR FOR
C         REGULAR DEPOSITS.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80
C         MODEL III.
C    ************************************
C    *    ORGANIZATION                  *
C    ************************************
C    INITIAL MESSAGE
C    PROBLEM PARAMETERS
C    HEADING
```

```
C    PROCESS
C    FINAL MESSAGE
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *     VARIABLES                                    *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    NYEARS    NUMBER OF YEARS
C    NPER      NUMBER OF PERIODS PER YEAR
C    IYEAR     CURRENT YEAR
C    IPER      CURRENT PERIOD
C    AMOUNT    AMOUNT OF REGULAR DEPOSIT
C    BAL       CURRENT BALANCE
C    YRATE     YEARLY RATE  (PERCENT)
C    PRATE     PERIOD RATE  (FRACTION)
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *     INITIAL MESSAGE                          *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         WRITE(2,100)
   100   FORMAT('1PROGRAM P0202'
      2              //' COMPUTE THE FUTURE VALUE'
      3               /' AT THE END OF EACH YEAR'
      4               /' FOR REGULAR DEPOSITS.')
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *     PROBLEM PARAMETERS                      *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         NYEARS  = 10
         NPER    = 12
         AMOUNT  = 50.0
         YRATE   = 12.75
         PRATE   = .01 * YRATE / NPER
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *     HEADING                                      *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         WRITE(2,200) AMOUNT, NPER, YRATE
   200   FORMAT(/' AMOUNT OF DEPOSIT PER PERIOD ',F10.2
      2             /' NUMBER OF PERIODS PER YEAR   ',I7
      3             /' ANNUAL INTEREST RATE         ',F13.5
      4             //' YEAR     BALANCE')
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *     PROCESS                                      *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         BAL = 0.0
         DO 390 IYEAR = 1, NYEARS
```

```
            DO 370  IPER = 1, NPER
                BAL = BAL * (1.0 + PRATE) + AMOUNT
      370       CONTINUE
                WRITE(2,380)  IYEAR,BAL
      380       FORMAT(I5,F10.2)
      390   CONTINUE
C     ************************************
C     *    FINAL MESSAGE                 *
C     ************************************
            WRITE(2,410)
      410   FORMAT(///' END OF OUTPUT')
            STOP
            END
```

Notice the indenting of the nested loops of this program. The inner loop is indented within the outer loop. For complex iterative programs involving deeply nested loops, indentation greatly helps readability. The output of this program is similar to that of the previous program.

2.3   Sequences

DEFINITION

A sequence is a set of values derived according to some rule. The sequence of numbers

$$1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7$$

is a simple example of a sequence. Each value constitutes one term of the sequence. The fifth term of the sequence is the value five.

MATHEMATICS

The field of mathematics is rich in the definition and application of sequences of numbers. The description of the behavior of physical phenomena over time and space may involve sequences of values. The future value of a savings account at the end of each year constitutes a term of a sequence.

COMPUTATION

The computer is a fast and accurate computational engine. The fastest computers can perform more calculations in one second

than a person could do in a lifetime. Computers play an important part in the calculation of sequences of numbers.

CONVERGENCE

Some sequences converge to a specifice value. As n increases, the nth term approaches the limiting value more closely. Some sequences are iterative in that the computational process must be done on a term-by-term basis. Other sequences allow the computation of the nth term directly without using the previous terms.

ESTIMATING THE CONSTANT e

The constant e is one of the most important constants in the field of mathematics. Its value is about 2.718. Like the constant pi, the constant e is irrational. The sequence defined by the expression

(1 + 1 / i) ** i

converges to the constant e for large values of i. This sequence converges very slowly. Modifying the sequence slightly produces faster convergence. The sequence defined by the expression

(1 + 1 / (10 ** i)) ** (10 ** i)

converges much more rapidly.

PROGRAM

The following program generates several terms of the sequence that converges to the constant e:

```
C   ***********************************
C   *    P0203                        *
C   ***********************************
C   AUTHOR
C        COPYRIGHT 1982
C        BY LAWRENCE MCNITT.
C   PURPOSE
C        ESTIMATE THE VALUE
C        FOR THE CONSTANT E
```

```
C          USING A SEQUENCE.
C     SYSTEM
C          MICROSOFT FORTRAN
C          RADIO SHACK TRS-80
C          MODEL III.
C     *************************************
C     *    ORGANIZATION                   *
C     *************************************
C     INITIAL MESSAGE
C     PROBLEM PARAMETERS
C     HEADING
C     PROCESS
C     FINAL MESSAGE
C     *************************************
C     *    VARIABLES                      *
C     *************************************
C     NUMBER     NUMBER OF TERMS
C     INDEX      CURRENT TERM
C     BEGIN      BEGINNING ARGUMENT
C     STEP       MULTIPLIER FOR NEXT ARGUMENT
C     VALUE      VALUE OF ARGUMENT
C     TERM       VALUE OF CURRENT TERM
C     *************************************
C     *    INITIAL MESSAGE                *
C     *************************************
          WRITE(2,110)
     110  FORMAT('1PROGRAM P0203'
     2               //' ESTIMATE THE VALUE OF'
     3               /' THE CONSTANT E USING'
     4               /' A SEQUENCE.')
C     *************************************
C     *    PROBLEM PARAMETERS             *
C     *************************************
          NUMBER = 10
          BEGIN  = 1.0
          STEP   = 2.0
          VALUE  = BEGIN
C     *************************************
C     *    HEADING                        *
C     *************************************
          WRITE(2,310)
     310  FORMAT(/' TERM      ESTIMATE')
```

```
C    ************************************
C    *    PROCESS                       *
C    ************************************
         DO 430 INDEX = 1, NUMBER
             VALUE = VALUE * STEP
             TERM = (1.0 + 1.0 / VALUE) ** VALUE
             WRITE(2,420) INDEX, TERM
     420     FORMAT(I5,F12.7)
     430 CONTINUE
C    ************************************
C    *    FINAL MESSAGE                 *
C    ************************************
         WRITE(2,510)
     510 FORMAT(///' END OF PROGRAM')
         STOP
         END
```

PRINTED OUTPUT

The following is the printed output produced by the program:

PROGRAM P0203

ESTIMATE THE VALUE OF
THE CONSTANT E USING
A SEQUENCE.

TERM	ESTIMATE
1	2.2500072
2	2.4414074
3	2.5657876
4	2.6379304
5	2.6769841
6	2.6973619
7	2.7076519
8	2.7127542
9	2.7156248
10	2.7172339

END OF PROGRAM

SEQUENCES FROM ELEMENTARY MATHEMATICS

Using the computer to generate the terms of sequences defined in elementary college mathematics courses is a productive endeavor. It provides practice in writing computer programs, and the output from running the programs aids in understanding the theory and application of mathematics.

2.4   Series

SUM OF A SEQUENCE

A series is the sum of the sequence. The arithmetic progression is a series. It consists of the sum of the values beginning with

$$1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7.$$

The sum of the first seven terms of the arithmetic progression

$$1 + 2 + 3 + 4 + 5 + 6 + 7$$

is 28.

CALCULUS

Calculus defines the Taylor's and Maclaurin's series. These are important in deriving series estimates of many functions. It is not necessary to understand the theory behind these series to use the computer to calculate series estimates.

CONVERGENCE

A series may converge to a limit just as a sequence may converge. One of the requirements for a series to converge to a finite value is that the corresponding sequence must converge to zero. The series estimate provides a method of approximating function values that would otherwise be more difficult to compute.

SERIES ESTIMATE OF THE CONSTANT e

The following series estimates the constant e:

$$1 + 1/1! + 1/2! + 1/3! + 1/4! + \ldots$$

This series converges to the constant *e* very rapidly. The term 4! means four factorial. This is the product of the first four integers. For positive integers *n*, the term *n*! is defined to be $n*(n-1)$! for $n = 1, 2, \ldots$. The value of 0! is defined to be one. The following table summarizes the factorials for the first few integers:

n	factorial
0	1
1	1
2	2
3	6
4	24
5	120

ITERATIVE PRODUCTS

The sum of a set of values is formed by accumulating them one at a time into an accumulator. Initially, the accumulator must be cleared, i.e., set to zero. Forming the product of a set of values requires a similar process. The initial value must be one rather than zero. The iterative process involves multiplying the previous product by the new term.

The following loop illustrates calculating *n*! (*n* factorial):

```
    FACT = 1.0
    DO 10 ITERM = 1, N
        FACT = FACT * ITERM
10  CONTINUE
```

This can be included in the routine calculating the series estimate of the constant *e*.

PROCEDURE

Consider using the previous series to estimate the constant *e*. The program can keep a running total of the series terms. It also maintains the sequence of terms by calculating the next term from the preceding term in an efficient manner.

PROGRAM

The following program calculates the terms of the sequence and
uses them to give the corresponding terms of the series:

```
C   ****************************************
C   *    P0204                             *
C   ****************************************
C   AUTHOR
C        COPYRIGHT 1982
C        BY LAWRENCE MCNITT.
C   PURPOSE
C        ESTIMATE THE VALUE
C        FOR THE CONSTANT E
C        USING A SERIES.
C   SYSTEM
C        MICROSOFT FORTRAN
C        RADIO SHACK TRS-80
C        MODEL III.
C   ****************************************
C   *    ORGANIZATION                      *
C   ****************************************
C   INITIAL MESSAGE
C   PROBLEM PARAMETERS
C   HEADING
C   PROCESS
C   FINAL MESSAGE
C   ****************************************
C   *    VARIABLES                         *
C   ****************************************
C   NUMBER    NUMBER OF TERMS
C   INDEX     CURRENT TERM
C   FACT      VALUE OF FACTORIAL
C   TERM      REAL VALUE OF TERM
C   SUM       SUM OF THE SERIES
C   ****************************************
C   *    INITIAL MESSAGE                   *
C   ****************************************
        WRITE(2,110)
    110 FORMAT('1PROGRAM P0204'
       2          //' ESTIMATE THE VALUE OF'
```

```
      3              /' THE CONSTANT E USING'
      4              /' A SERIES.')
C   *****************************************
C   *     PROBLEM PARAMETERS              *
C   *****************************************
        NUMBER = 10
        TERM   = 0.0
        FACT   = 1.0
        SUM    = 1.0
C   *****************************************
C   *     HEADING                         *
C   *****************************************
        WRITE(2,310)
310     FORMAT(/' TERM    ESTIMATE')
C   *****************************************
C   *     PROCESS                         *
C   *****************************************
        DO 430 INDEX = 1, NUMBER
            TERM = TERM + 1.0
            FACT = FACT * TERM
            SUM  = SUM + 1.0 / FACT
            WRITE(2,420) INDEX, SUM
420         FORMAT(I5,F12.7)
430     CONTINUE
C   *****************************************
C   *     FINAL MESSAGE                   *
C   *****************************************
        WRITE(2,510)
510     FORMAT(///' END OF PROGRAM')
        STOP
        END
```

PRINTED OUTPUT

The following printed output results from the program giving the series estimates of e for the first few terms:

PROGRAM P0204

ESTIMATE THE VALUE OF
THE CONSTANT E USING
A SERIES.

TERM	ESTIMATE
1	2.0000000
2	2.5000000
3	2.6666667
4	2.7083335
5	2.7166669
6	2.7180557
7	2.7182541
8	2.7182789
9	2.7182817
10	2.7182820

END OF PROGRAM

2.5 Round-off Errors

PRECISION

Single precision real variables contain the equivalent of seven decimal digits of precision. Seven significant digits are adequate for most applications. Double precision real variables contain the equivalent of 16 significant digits. They are sufficient for most critical applications requiring high precision.

FINITE PRECISION

Computers provide finite precision. Many values are approximated to the stated number of significant digits. Small errors may be introduced during the conversion to internal form. Further errors may accumulate during the course of the calculations.

ERRORS INTRODUCED BY CALCULATION

Significant errors may result from aligning the values during the calculation. Alignment errors are most evident in addition and subtraction. For numbers of widely different magnitude the computer aligns the decimal points before the operation. This can result in the loss of significant digits.

The problem is worst when subtracting values that are almost identical. Consider subtracting the value 456.1285 from

the value 456.1297. The difference is .0012. Each value had seven significant digits but the result has only two significant digits.

Careful consideration should be given to the needs for precision for the application and its calculations. Double precision variables require twice the storage space but give more than twice the precision of single precision variables. Double precision arithmetic takes longer than single precision arithmetic.

ESTIMATING THE SLOPE OF A FUNCTION

In calculus, the derivative is a function giving the slope of the tangent line to another function at every point. The derivative of a function at a point gives the slope of the tangent line at that point. Let X be the point of interest. Let X' be a second point very close to the value of X. Let Y be the value of the function at X and Y' be the value at the point X'. The expression

$$(Y' - Y) / (X' - X)$$

estimates the slope of the function at the point X.

This can be described in terms of a limiting process. Moving the point X' closer and closer to X results in an estimate that converges to the true slope. This assumes that the function has the necessary properties required for the slope to be defined. The function must be defined at the point in question and be continuous in the neighborhood of that point. Certain other necessary properties are given in elementary calculus.

This is a classic problem illustrating round-off errors. Both the numerator and the denominator involve subtractions of values that are nearly equal. Consider the function

$$Y = f(X) = X * X$$

giving Y as the square of X. This function has the slope 1 at the point $X = 2$.

The program in this section tries the sequence of values 2.1, 2.01, 2.001, . . . for the X' for making the estimate. The estimate of the slope improves for the first few terms. After that the estimate gets slightly worse for several terms and then the estimate becomes meaningless. This results from the seven significant-digits limit of single precision variables.

Double precision variables would allow better estimates of the slope at a point. Eventually problems will develop for any level of precision. The choice for the limits for estimating the slope depends on the magnitude of the value of the function and on the precision of the number representation.

PROGRAM

The following program estimates the value of the slope at the point $X = 2$ for several limits:

```
C    *************************************
C    *     P0205                          *
C    *************************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         ESTIMATE THE SLOPE
C         OF A FUNCTION
C         AT A POINT.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80
C         MODEL III.
C    *************************************
C    *     ORGANIZATION                   *
C    *************************************
C    INITIAL MESSAGE
C    PROBLEM PARAMETERS
C    HEADING
C    PROCESS
C    FINAL MESSAGE
C    *************************************
C    *     VARIABLES                      *
C    *************************************
C    NUMBER    NUMBER OF TERMS
C    INDEX     CURRENT TERM
C    WIDTH     WIDTH OF INTERVAL
C    X1        FIRST ARGUMENT
C    X2        SECOND ARGUMENT
```

```
C     Y1           VALUE OF FUNCTION AT FIRST ARGUMENT
C     Y2           VALUE OF FUNCTION AT SECOND ARGUMENT
C     SLOPE        ESTIMATE OF SLOPE
C     **************************************
C     *     INITIAL MESSAGE                 *
C     **************************************
          WRITE(2,110)
   110    FORMAT('1PROGRAM P0205'
      2             //' ESTIMATE THE SLOPE'
      3             /' OF A FUNCTION'
      4             /' AT A POINT.')
C     **************************************
C     *     PROBLEM PARAMETERS              *
C     **************************************
          NUMBER = 12
          WIDTH  = 1.0
          X1     = .5
          Y1     = X1 * X1
C     **************************************
C     *     HEADING                         *
C     **************************************
          WRITE(2,310)
   310    FORMAT(/'          WIDTH        SLOPE')
C     **************************************
C     *     PROCESS                         *
C     **************************************
          DO 430 INDEX = 1, NUMBER
              WIDTH = WIDTH / 10.0
              X2    = X1 + WIDTH
              Y2    = X2 * X2
              SLOPE = (Y2 - Y1) / WIDTH
              WRITE(2,420) WIDTH, SLOPE
   420        FORMAT(2F15.12)
   430    CONTINUE
C     **************************************
C     *     FINAL MESSAGE                   *
C     **************************************
          WRITE(2,510)
   510    FORMAT(///' END OF PROGRAM')
          STOP
          END
```

PRINTED OUTPUT

The following printed output results from the program estimating the slope at the point $X = 2$:

PROGRAM P0205

ESTIMATE THE SLOPE
OF A FUNCTION
AT A POINT.

WIDTH	SLOPE
.100000001490	1.100000143051
.009999999776	1.009997725487
.000999999931	1.001000523567
.000099999990	1.000166058540
.000009999999	1.001358151436
.000001000000	1.013279080391
.000000100000	1.192093014717
.000000010000	0.000000000000
.000000001000	0.000000000000
.000000000100	0.000000000000
.000000000010	0.000000000000
.000000000001	0.000000000000

END OF PROGRAM

2.6   Exercises

1. Write a program calculating the distance traveled in feet at the end of each of the first 20 seconds for an object accelerating at the constant rate of eight feet/sec/sec.

2. Write a program calculating the sum of the squares of the first 25 integers.

3. Modify the program estimating the slope of the function

$$Y = f(X) = X ** 2$$

to use double precision. What interval width gives the best estimate for the slope at the point $X = 2$?

4. Write a program computing the future value of $1,275.25 at the end of each of the first five years assuming an interest rate of 12.75 percent compounded quarterly.

5. The sequence containing the nth term

$$(1 + 1 / N) ** N$$

converges to the constant e as N increases. Write a program estimating the constant e using the values N = 10, 20, 30, . . ., 150.

6. Elementary calculus includes differential calculus and integral calculus. One interpretation for a definite integral is as the area under a curve. Let

$$Y = f(X) = X ** 2$$

be the function in question. Use numerical integration to estimate the area under the curve defined by the function between the values X = 1.5 and X = 2.5. Subdivide the interval between X = 1.5 and X = 2.5 into 100 subintervals. Evaluate $f(X)$ at the midpoint of each subinterval. The subinterval times the value of $f(X)$ gives an estimate of the area under the curve within that subinterval. The sum of the areas defined for all 100 subintervals estimates the area under the curve between the limits X = 1.5 and X = 2.5.

3 Conditionals

OVERVIEW Conditional statements permit the computer to be flexible. Not all problem situations need exactly the same analysis. Some situtations require one set of instructions while other situations require other approaches.

The IF statement is the primary conditional command. The traditional method has been to employ IF . . . GO TO instructions. The resulting branches are hard to follow. Recent versions of FORTRAN include more general IF statements that do not involve branching.

☐

3.1 Generalized Conditional

GENERALIZED IF

The generalized IF statement has a condition and an executable statement. If the condition is true, the computer executes the statement portion. If the condition is false, the computer does not execute the statement portion.

CONDITION

The condition is contained within the parentheses immediately following the IF keyword. The condition is a logical expression. The expression

(A.LT.B)

returns the value "1" for true or "0" for false.

51

The following lists the relational operators for logical expressions:

.LT. Less than
.LE. Less than or equal to
.EQ. Equal to
.NE. Not equal to
.GE. Greater than or equal to
.GT. Greater than

IF STATEMENTS

The IF statement

IF (A.LT.B) SUM = SUM + VALUE

adds the contents of the variable VALUE to the variable SUM if the condition is true. The statement following the condition can be any executable statement except the DO statement or another logical IF statement. The command

IF (VAL.EQ.2.0) WRITE(2,210) A, B, C

executes the WRITE command if the variable VAL contains the value 2.0.

CALCULATED SUBEXPRESSIONS

The condition may include complex calculations along with the relation operator. The command

IF ((INDEX/NUMBER) .GT. (14+6*J)) KS=KS+1

uses expressions as part of the condition test.

LOGICAL EXPRESSIONS

Logical expressions consisting of "AND", "OR", "NOT", and "XOR" are also possible within IF statements. The command

IF (A.LT.B.AND.L.EQ.M) READ(6,124) A, B

performs the READ only if both conditions are true. The "XOR" operator is the exclusive or. The result is true if one of the logical operands is true but not if both are true. The "OR" operator is the inclusive or. The result is true if at least one of the operands is true.

READABILITY

FORTRAN follows a precedence ordering for complex logical expressions. Arithmetic operations take precedence over relational operations which take precedence over logical operations. Parentheses help avoid confusion. The command

IF ((A.LT.B) .AND. (L.EQ.M)) READ(6,124) A. B

illustrates this.

METRIC TO ENGLISH CONVERSION

The program of this section converts meters to feet and inches. It illustrates the use of conditionals by printing the values for feet or inches or both. A value of zero for either measure inhibits its printing.

PROGRAM

The following program uses conditional statements in its OUT-PUT section:

```
C    ************************************
C    *     P0301                              *
C    ************************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         METRIC TO ENGLISH
C         CONVERSION.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80
```

```
C          MODEL III.
C    *************************************
C    *     ORGANIZATION                     *
C    *************************************
C    INITIAL MESSAGE
C    PROBLEM PARAMETERS
C    PROCESS
C    OUTPUT
C    FINAL MESSAGE
C    *************************************
C    *     VARIABLES                        *
C    *************************************
C    XMETER    MEASUREMENT IN METERS
C    FACTOR    CONVERSION FACTOR
C    IFEET     NUMBER OF FEET
C    XINCH     NUMBER OF INCHES
C    *************************************
C    *     INITIAL MESSAGE                   *
C    *************************************
          WRITE(1,110)
    110   FORMAT(///' PROGRAM P0301'
        2           //' CONVERT LENGTH IN METERS'
        3            /' TO FEET AND INCHES.')
C    *************************************
C    *     PROBLEM PARAMETERS                *
C    *************************************
          FACTOR = 39.37
          XMETER = 3.47
C    *************************************
C    *     PROCESS                           *
C    *************************************
          XINCH = FACTOR * XMETER
          IFEET = XINCH / 12.0
          XINCH = XINCH - 12.0 * IFEET
C    *************************************
C    *     OUTPUT                            *
C    *************************************
          WRITE(1,310) XMETER
    310   FORMAT(/' METERS = ',F10.5/)
          IF (IFEET.GT.0) WRITE(1,320) IFEET
    320   FORMAT(' FEET    = ',I5)
          IF (XINCH.GT.0.0) WRITE(1,330) XINCH
```

```
     330    FORMAT(' INCHES = ',F10.5)
C    *********************************************
C    *    FINAL MESSAGE                          *
C    *********************************************
         WRITE(1,410)
     410    FORMAT(/' END OF PROGRAM'/)
            STOP
            END
```

TEST RUN

The following shows the video output for the test run with the program:

PROGRAM P0301

CONVERT LENGTH IN METERS
TO FEET AND INCHES.

METERS =   3.47000

FEET   =   11
INCHES =   4.61389

END OF PROGRAM

3.2   GO TO

BRANCHES

A branch is a transfer of control from one place in the program to another place. The simplest command is the unconditional branch. The statement

GO TO 420

is typical. The program transfers control to the statement numbered 420.

CONDITIONAL BRANCH

The conditional branch transfers control only if the condition is true. The command

IF (A.LT.B) GO TO 420

transfers control to the line numbered 420 only if the value of *A* is less than the value of *B*.

PRIMITIVE BRANCH

The conditional and unconditional branch statements are primitive branches. They are primitive in the sense that they can go anywhere within the program. The change is abrupt. The complete lack of control makes branches hard to understand. The program becomes a complex maze. Some liken the program to a bowl of spaghetti. Following the twisted paths through the program is almost impossible.

CONTROLLED BRANCHES

Branches are necessary for some situations but they do not have to be primitive, uncontrolled branches. The Do-loop mechanism is a prime example of a controlled branching mechanism. The DO statement contains a statement number identifying the last statement of the loop. The DO statement itself calls the programmer's attention to the fact that a loop is involved.

The last statement of the loop includes an implied branch back to the beginning of the loop if further iterations are necessary. The Do-loop is easier to understand than alternative methods using primitive GO TO and IF . . . GO TO statements.

ELIMINATION OF THE GO TO

If the language includes certain commands, it is possible to write any program without primitive branches. The necessary language constructs include a subroutine mechanism, a comprehensive IF . . . THEN . . . ELSE construct, and a DO-WHILE construct. The ultimate goal is readability, not the elmination of all GO TO statements.

FORTRAN AND THE GO TO

FORTRAN includes the computed Do-loop mechanism which is more specific than the DO-WHILE. Most FORTRAN compilers, including the one by MICROSOFT, do not include the compre-

hensive IF . . . THEN . . . ELSE construct. This is probably the greatest limitation of existing versions of FORTRAN.

The latest FORTRAN standards do include these language constructs and future compilers will begin implementing these features. This illustrates the evolutionary upgrading of FORTRAN during the last 25 years and is one of the reasons why FORTRAN survives as an applications language.

3.3   IF . . . GO TO

CONDITIONAL BRANCH

The conditional branch follows the GO TO path if the condition is true. The statement

IF (A.LT.B) GO TO 420

branches to the statement numbered 420 if the condition is true. The conditional branch is a primitive branch just like the unconditional GO TO statement.

READABILITY

The GO TO and the IF . . . GO TO statements can result in unreadable programs. Programs are usually more readable if they use fewer primitive branches. Programmers should use controlled branching mechanisms such as the Do-loop and the subroutine call.

GENERALIZED IF

The generalized IF statement provides flexibility without branches. As newer compilers incorporate more comprehensive IF statements this statement will increase in importance. This is one of the most productive uses of the IF statement.

ARITHMETIC IF

The arithmetic IF statement is discouraged because of its three-way branch. It includes three statement numbers and an integer

expression within parentheses. If the expression is negative, the branch is to the first statement number. If the expression is zero, the branch is to the second statement number. If the expression is positive, the branch is to the third statement number. The statement

IF (K-2*L) 400,500,785

illustrates the arithmetic IF command.

CASE SELECTION

A problem may require special handling for each of several situations. Each case requires its own processing methods. One practice is to define a variable to designate the case and code that variable, letting the value 1 represent case 1, the value 2 represent case 2, etc.

A series of IF . . . GO TO statements distribute the program flow of control to the sections of the program handling the cases. The following sequence of commands accomplishes this:

```
IF (ICASE.EQ.1) GO TO 1000
IF (ICASE.EQ.2) GO TO 1100
IF (ICASE.EQ.3) GO TO 1200
IF (ICASE.EQ.4) GO TO 1300
IF (ICASE.EQ.5) GO TO 1400
```

This may be tedious but the resulting program flow is evident.

After each case is processed, control flows back to a common collection point. Although these are primitive branches, they are proper if used in a very controlled fashion. This control is important in the design of readable programs.

COMPUTED GO TO

The computed GO TO statement provides a shortened statement accomplishing the same thing as the sequence of IF statements illustrated above. The statement

GO TO (1000,1100,1200,1300,1400) ICASE

selects the statement depending on the value of the variable ICASE. If the variable contains the value 1, control goes to the first statement number. If the value is 2, control passes to the second statement number, etc.

If the variable contains a value less than one or greater than the number of statement numbers within the parentheses, control falls through to the statement following the computed GO TO. If necessary, this contains an error routine.

PROGRAM

The program of this section illustrates the problems that result from extensive use of GO TO and IF . . . GO TO statements. It comes from Chapter 2 and computes the accumulated value of an annuity at the end of each of several years, assuming monthly deposits.

The program implements the loops using primitive branches instead of using nested Do-loops. The PROCESS section of the program is much more difficult to understand than the equivalent section using the DO mechanism. The program follows:

```
C    **************************************
C    *    P0303                          *
C    **************************************
C    AUTHOR
C        COPYRIGHT 1982
C        BY LAWRENCE MCNITT.
C    PURPOSE
C        FUTURE VALUE AT END
C        OF EACH YEAR FOR
C        REGULAR DEPOSITS.
C    SYSTEM
C        MICROSOFT FORTRAN
C        RADIO SHACK TRS-80
C        MODEL III.
C    **************************************
C    *    ORGANIZATION                   *
C    **************************************
C    INITIAL MESSAGE
C    PROBLEM PARAMETERS
```

```
C     HEADING
C     PROCESS
C     FINAL MESSAGE
C     ***********************************
C     *     VARIABLES                   *
C     ***********************************
C     NYEARS    NUMBER OF YEARS
C     NPER      NUMBER OF PERIODS PER YEAR
C     IYEAR     CURRENT YEAR
C     IPER      CURRENT PERIOD
C     AMOUNT    AMOUNT OF REGULAR DEPOSIT
C     BAL       CURRENT BALANCE
C     YRATE     YEARLY RATE  (PERCENT)
C     PRATE     PERIOD RATE  (FRACTION)
C     ***********************************
C     *     INITIAL MESSAGE             *
C     ***********************************
          WRITE(2,100)
     100      FORMAT('1PROGRAM P0303'
     2               //' COMPUTE THE FUTURE VALUE'
     3               /' AT THE END OF EACH YEAR'
     4               /' FOR REGULAR DEPOSITS.')
C     ***********************************
C     *     PROBLEM PARAMETERS          *
C     ***********************************
          NYEARS = 10
          NPER   = 12
          AMOUNT = 50.0
          YRATE  = 12.75
          PRATE  = .01 * YRATE / NPER
C     ***********************************
C     *     HEADING                     *
C     ***********************************
          WRITE(2,200) AMOUNT, NPER, YRATE
     200      FORMAT(/' AMOUNT OF DEPOSIT PER PERIOD ',F10.2
     2               /' NUMBER OF PERIODS PER YEAR   ',I7
     3               /' ANNUAL INTEREST RATE         ',F13.5
     4               //' YEAR   BALANCE')
C     ***********************************
C     *     PROCESS                     *
C     ***********************************
```

```
        BAL = 0.0
        IYEAR = 0
310     IYEAR = IYEAR + 1
        IF (IYEAR.GT.NYEARS) GO TO 390
        IPER = 0
320     IPER = IPER + 1
        IF (IPER.GT.NPER) GO TO 370
        BAL = BAL * (1.0 + PRATE) + AMOUNT
        GO TO 320
370     WRITE(2,380) IYEAR,BAL
380     FORMAT(I5,F10.2)
        GO TO 310
390     CONTINUE
C  ***********************************
C  *    FINAL MESSAGE                *
C  ***********************************
        WRITE(2,410)
410     FORMAT(///' END OF OUTPUT')
        STOP
        END
```

PRINTED OUTPUT FROM TEST RUN

The following is the printed output produced by the test run:

PROGRAM P0303

COMPUTE THE FUTURE VALUE
AT THE END OF EACH YEAR
FOR REGULAR DEPOSITS.

AMOUNT OF DEPOSIT PER PERIOD	50.00
NUMBER OF PERIODS PER YEAR	12
ANNUAL INTEREST RATE	12.75000

YEAR	BALANCE
1	636.33
2	1358.71
3	2178.78
4	3109.73
5	4166.56
6	5366.31
7	6728.28

```
      8         8274.42
      9        10029.63
     10        12022.18
```

END OF OUTPUT

3.4 Conditional Termination of a Loop

LACK OF DO-WHILE

Future versions of FORTRAN will include the generalized DO-WHILE construct. The concern of this book is with the version of Microsoft's FORTRAN implemented in Radio Shack computers.

One approach for present versions involves initiating a computed Do-loop with a very large number of iterations. An IF . . . GO TO from within the loop branches out of the loop when the appropriate condition is sensed. Some call this a DO-forever command. This method may be confusing because there is no intention of repeating the Do-loop the specified number of times. A conditional branch is used to break out of the loop.

SERIES ESTIMATE OF THE CONSTANT e

The program of this section uses the series estimate of the constant e from the previous chapter. The program of that chapter performed the Do-loop a predetermined number of times. The approach of this section is to perform the loop until the last term of the summation is suitably small.

PROGRAM

The following program continues adding terms to the series estimate of the constant e until the last term is less than a predefined limit:

```
C    ***********************************
C    *     P0304                       *
C    ***********************************
C    AUTHOR
C          COPYRIGHT 1982
```

```
C          BY LAWRENCE MCNITT.
C     PURPOSE
C          ESTIMATE THE VALUE
C          FOR THE CONSTANT E
C          USING A SERIES.
C     SYSTEM
C          MICROSOFT FORTRAN
C          RADIO SHACK TRS-80
C          MODEL III.
C     ************************************
C     *      ORGANIZATION                *
C     ************************************
C     INITIAL MESSAGE
C     PROBLEM PARAMETERS
C     HEADING
C     PROCESS
C     FINAL MESSAGE
C     ************************************
C     *      VARIABLES                   *
C     ************************************
C     NUMBER    NUMBER OF TERMS
C     INDEX     CURRENT TERM
C     FACT      VALUE OF FACTORIAL
C     TERM      REAL VALUE OF TERM
C     CHECK     CHECK FOR EXIT FROM LOOP
C     SUM       SUM OF THE SERIES
C     ************************************
C     *      INITIAL MESSAGE             *
C     ************************************
          WRITE(2,110)
      110    FORMAT('1PROGRAM P0304'
        2            //' ESTIMATE THE VALUE OF'
        3            /' THE CONSTANT E USING'
        4            /' A SERIES.')
C     ************************************
C     *      PROBLEM PARAMETERS          *
C     ************************************
          NUMBER = 1000
          TERM   = 0.0
          FACT   = 1.0
          SUM    = 1.0
          CHECK  = .0001
```

```
C     ************************************
C     *    HEADING                       *
C     ************************************
      WRITE(2,310)
310   FORMAT(/' TERM      ESTIMATE')
C     ************************************
C     *    PROCESS                       *
C     ************************************
      DO 430  INDEX = 1, NUMBER
         TERM = TERM + 1.0
         FACT = FACT * TERM
         SUM  = SUM + 1.0 / FACT
         WRITE(2,420) INDEX, SUM
420      FORMAT(I5,F12.7)
         IF  (CHECK.GT.(1.0/FACT) ) GO TO 440
430   CONTINUE
440   CONTINUE
C     ************************************
C     *    FINAL MESSAGE                 *
C     ************************************
      WRITE(2,510)
510   FORMAT(///' END OF PROGRAM')
      STOP
      END
```

PRINTED OUTPUT

The following shows the printed output generated by the program:

PROGRAM P0304

ESTIMATE THE VALUE OF
THE CONSTANT E USING
A SERIES.

TERM	ESTIMATE
1	2.0000000
2	2.5000000
3	2.6666667
4	2.7083335
5	2.7166669
6	2.7180557

```
7      2.7182541
8      2.7182789
```

END OF PROGRAM

3.5   Exercises

1. Write a program listing all the integers that divide evenly into the value 12456.

2. Write a program listing all the prime factors of the value 12456.

3. Write a program using a series to estimate the value of $\sin(X)$ for $X = .415$ in radians. Terminate when the last term is less than .00001. An elementary calculus textbook will discuss the series estimate approach.

4. Write a program computing gross pay from the hourly pay rate and the number of hours worked. Include time-and-a-half for more than 40 hours.

5. Write a program that will search for the value X for which $f(X)$ = 0. Use the method known as bisection. Use the function

$$f(X) = X*X + 2*X - 3.$$

The value of $f(X)$ for $X = 1.5$ is 2.25. The value of $f(X)$ for $X = 0 = -3$. The function $f(X)$ should become zero at least once in the interval $X = .5$ to $X = 1.5$. This assumes that the function is defined and is continuous throughout the region of interest. Evaluate $f(X)$ at the midpoint between 0 and 1.5. If the value for $f(X)$ is suitably close to zero, say .00001, terminate the processing. Otherwise, let the midpoint become the new boundary and inspect the value $f(X)$ for the new midpoint.

6. Estimate the value of X for which the function

$$f(X) = X*X - 2*X + 3$$

is minimized. Search the points $X = 4, -3.99, -3.98, \ldots, 4$. Print the value of X and the value of $f(X)$ for the minimum.

4 Built-in functions

OVERVIEW The FORTRAN Compiler makes a large number of built-in functions available for incorporation into the program. These include trigonometric functions, logarithmic functions, the square root function, and many general-purpose functions.

☐

4.1 SQRT, ABS, SIGN

DEFINITION

A function is a built-in sequence of commands referenced by name. It consists of the function name followed by one or more data items enclosed in parentheses. A call to the function returns a value which can be used as any data value in a FORTRAN expression.

SQRT

The SQRT function returns the square root of the argument enclosed in parentheses. The expression

SQRT (25.0)

returns the square root of the value 25. The argument must be a single precision real value; it cannot be integer or double precision. The expression

SQRT (VALUE)

returns the square root of the contents of the variable VALUE.

67

USE OF FUNCTIONS

The result of the function can be used as any value in other expressions. The statement

SQROOT = SQRT(VALUE)

places the square root of the variable VALUE into the variable SQROOT. The statement

DATA = 5.75 * SQRT(VALUE)

uses the result of the function in a larger expression. The statement

D1 = SQRT(PERT/100.0+START)

returns the square root of the result of the expression within parentheses.

DATA TYPING

FORTRAN built-in functions presume the data type implied by the function name. Function names beginning with the letters *I* through *N* return integer results. Functions having names beginning with other letters return single precision real values.

DOUBLE PRECISION

Functions having names beginning with the letter *D* are usually double precision functions returning double precision results. The argument is usually double precision as well.
     The statement

DSQRT(2.0D2)

gives the double precision square root of the value 200. The argument to DSQRT must also be double precision. Double pre-

cision literals must be in scientific notation with the *D* symbol identifying the exponent portion.

SIGN

The SIGN function for Microsoft FORTRAN requires two arguments separated by commas. The function returns a value having the value of the first argument and the sign from the second argument.

There are three SIGN functions. ISIGN requires integer arguments and returns an integer result. DSIGN requires double precision arguments and returns a double precision result. SIGN requires single precision real arguments and returns a single precision real result.

The statement

VALUE = SIGN(1.0,-25.75)

places the value −1.0 into the variable called VALUE. VALUE must be single precision. The statement

DATA = DSIGN(DATA,S11)

makes the variable called DATA have the same sign as the variable called S11. The result and the arguments must be explicitly defined as double precision. The statement

IVAL = ISIGN(144,IDATA)

places the value 144 into the variable IVAL using the sign of the value in IDATA.

ABS

The ABS function returns the absolute value of the argument. The ABS function returns a single precision result and assumes a single precision argument. The DABS function returns a double precision result and requires a double precision argument. The IABS

function returns an integer result and requires an integer argument. The statement

XMAGN = ABS(VALUE)

returns the single precision absolute value of the single precision argument VALUE.

NUMBER REPRESENTATIONS

These functions typify FORTRAN built-in functions available for incorporation into programs. They are an important part of the FORTRAN compiler and simplify the programming process. They illustrate the emphasis that FORTRAN places upon number representations. This is a curse to the beginning programmer but is a blessing to the experienced programmer in the control that it gives over storage efficiency, precision, and processing speed.

PROGRAM GENERATING SQUARE ROOT TABLE

The following program uses the SQRT function in the process of generating a square root table:

```
C    ************************************
C    *    P0401                         *
C    ************************************
C    AUTHOR
C        COPYRIGHT 1982
C        BY LAWRENCE MCNITT.
C    PURPOSE
C        GENERATE SQUARE
C        ROOT TABLE.
C    SYSTEM
C        MICROSOFT FORTRAN
C        RADIO SHACK TRS-80
C        MODEL III.
C    ************************************
C    *    ORGANIZATION                  *
C    ************************************
C    INITIAL MESSAGE
C    PROBLEM PARAMETERS
C    PROCESS
```

```
C    FINAL MESSAGE
C    ***********************************
C    *     VARIABLES                    *
C    ***********************************
C    NUMBER    NUMBER OF VALUES
C    ICOUNT    COUNTER FOR VALUES
C    VALUE     CURRENT VALUE
C    STEP      STEP SIZE FOR VALUES
C    SQROOT    SQUARE ROOT
C    ***********************************
C    *     INITIAL MESSAGE              *
C    ***********************************
          WRITE(1,110)
    110   FORMAT(/' PROGRAM P0401'
      2            //' GENERATE SQUARE'
      3             /' ROOT TABLE.')
          WRITE(2,120)
    120   FORMAT('1SQUARE ROOT TABLE'
      2            //5X, 'VALUE' ,3X, 'SQUARE ROOT')
C    ***********************************
C    *     PROBLEM PARAMETERS           *
C    ***********************************
          VALUE  = 1.0
          STEP   = 0.1
          NUMBER = 10
C    ***********************************
C    *     PROCESS                      *
C    ***********************************
          DO 320 ICOUNT = 1, NUMBER
              SQROOT = SQRT(VALUE)
              WRITE(2,310) VALUE, SQROOT
    310       FORMAT(2F10.5)
              VALUE = VALUE + STEP
    320   CONTINUE
C    ***********************************
C    *     FINAL MESSAGE                *
C    ***********************************
          WRITE(2,410)
    410   FORMAT(///' END OF OUTPUT')
          WRITE(1,420)
    420   FORMAT(/' END OF PROGRAM')
          STOP
          END
```

PRINTED OUTPUT FROM TEST RUN

The following gives the printed output produced by the test run:

SQUARE ROOT TABLE

VALUE	SQUARE ROOT
1.00000	1.00000
1.10000	1.04881
1.20000	1.09545
1.30000	1.14018
1.40000	1.18322
1.50000	1.22474
1.60000	1.26491
1.70000	1.30384
1.80000	1.34164
1.90000	1.37840

4.2 Exponential and Logarithmic

LOGARITHMS

A logarithm is an exponent. The base ten logarithm of the value X is the exponent B for the value 10 which gives the resulting value X. In the FORTRAN expression

$$X = 10.0 ** B$$

the value in B is the base ten logarithm of the resulting value in X. Ten raised to the Bth power is X, therefore, B is the base ten logarithm of X.

For this to be true the value in X must be positive. A positive value for B results in a value of X greater than the value 1.0. A negative value for B results in a value of X falling between 0.0 and 1.0.

ANTILOG

The antilog reverses the logarithm process. If B is the logarithm of X, then X is the antilog of B. Given a logarithm B, raising the value 10.0 to that power gives the antilog.

OVERFLOW AND UNDERFLOW

Microsoft FORTRAN on the Radio Shack TRS-80 represents real values having a magnitude between $10**-38$ and $10**38$. Attempting to represent any value greater than $10**38$ results in overflow. Attempting to represent any value closer to zero than $10**-38$ results in underflow.

CALCULATION WITH LOGARITHMS

Calculation with logarithms reduces the risk of overflow and underflow for some problems. Rather than form the product of a series of terms, form the sum of their logarithms. Rather than raise the value A to the nth power, compute n times the logarithm of A.

TRANSLATION

The final result of the calculation using logarithms is the logarithm of the desired answer. If the answer does not overflow or underflow, then raising the value 10.0 to the power given by the final logarithm gives the desired decimal answer.

  If overflow or underflow is possible, then the antilog requires a two-step process. The integer part of the final logarithm gives the exponent of the base ten result. The fractional part becomes the base ten exponent for the mantissa of the result. The program of this section uses this approach to avoid overflow problems.

LOG FUNCTIONS

The LOG functions include ALOG, DLOG, ALOG10, and DLOG-10. ALOG is a single precision function returning the natural logarithm of the argument. DLOG is a double precision function returning the natural (base e) logarithm of the argument. ALOG10 is a single precision function returning the common (base ten) logarithm of the argument. DLOG10 is a double precision function returning the common logarithm of the argument.

EXP FUNCTIONS

The EXP functions return the exponential value of the argument. These functions raise the value *e* to the power given by the argument. This corresponds to finding the antilog of base ten logarithms. EXP is a single precision function raising *e* to the power specified by the argument. DEXP is a double precision function raising *e* to the power given by the argument.

FACTORIALS

The value *n*! (called *n* factorial) is the product of the first *n* integers if *n* is greater than zero. The value of 0! is defined to be 1.

INTEGER CALCULATIONS

Integer calculations seem natural for factorials. The following FORTRAN segment computes *n*!:

```
      IFACT = 1
      DO 110 ICOUNT = 1, N
          IFACT = IFACT * ICOUNT
110   CONTINUE
```

This routine works for small nonzero values of *n*. Factorials become very large for large *n*. Integers are inadequate for representing values of large magnitude.

REAL NUMBER REPRESENTATION

The real number representation allows values up to 10**38. Although this is adequate for most problems, it still does not handle *n*! for large *n*. The following FORTRAN program segment uses real numbers to calculate *n*!:

```
      FACT = 1.0
      DO 110 ICOUNT = 1, N
          TERM = ICOUNT
          FACT = FACT * TERM
110   CONTINUE
```

The statement

TERM = ICOUNT

converts the integer ICOUNT to the real variable TERM.
Most FORTRAN compilers will accept the statement

FACT = FACT * ICOUNT

which includes mixed data types in the expression to the right
of the equals sign. This is not good practice, but it works. The
computer converts integer values in mixed-mode expressions to
real format before doing the calculations.

A better approach is to use the FLOAT function which
returns the real form of the integer argument. The following
FORTRAN segment uses the FLOAT function:

```
      FACT = 1.0
      DO 110 ICOUNT = 1, N
           FACT = FACT * FLOAT(ICOUNT)
110   CONTINUE
```

The reason why this is a better approach is that it signals the data
conversion that must take place.

CALCULATION WITH LOGARITHMS

Overflow is still a severe problem with real numbers. The value for
65! will overflow, for example. Calculating with logarithms in-
volves summing the logarithms of the first n integers rather than
forming the product of those integers. The result is the logarithm
of the factorial. Further calculations then use the log factorial.

The following program segment computes the base ten log
factorial:

```
      FACT = 0.0
      DO 110 ICOUNT = 1, N
           TERM = ICOUNT
           FACT = FACT + ALOG10(TERM)
110   CONTINUE
```

FACTORIAL TABLE FOR LARGE *N*

The following program generates a factorial table for large *N*, extracting the exponent and mantissa and printing the two parts separately:

```
C    *********************************
C    *    P0402                       *
C    *********************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         GENERATE FACTORIAL TABLE
C         USING LOGARITHMS.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80
C         MODEL III.
C    *********************************
C    *    ORGANIZATION                *
C    *********************************
C    INITIAL MESSAGE
C    PROBLEM PARAMETERS
C    PROCESS
C    FINAL MESSAGE
C    *********************************
C    *    VARIABLES                   *
C    *********************************
C    ICOUNT    COUNTER FOR VALUES
C    ISTART    STARTING POINT
C    ISTOP     STOPPING POINT
C    VALUE     CURRENT VALUE
C    FLOG      LOG FACTORIAL
C    IEXP      EXPONENT
C    FRACT     MANTISSA
C    *********************************
C    *    INITIAL MESSAGE             *
C    *********************************
         WRITE(1,110)
   110   FORMAT(/' PROGRAM P0402'
      2          //' GENERATE FACTORIAL TABLE'
```

```
      3              /' USING LOGARITHMS.')
          WRITE(2,120)
   120    FORMAT('1FACTORIAL TABLE'
      2              //5X, 'VALUE' ,3X, 'FACTORIAL')
C   ***********************************
C   *    PROBLEM PARAMETERS           *
C   ***********************************
          ISTART = 100
          ISTOP  = 110
          VALUE  = 0.0
          FLOG   = 0.0
C   ***********************************
C   *    PROCESS                      *
C   ***********************************
          DO 320 ICOUNT = 1, ISTOP
              VALUE  = FLOAT(ICOUNT)
              FLOG   = FLOG + ALOG10(VALUE)
              IF (ICOUNT.LT.ISTART) GO TO 320
              IEXP    = FLOG
              FRACT  = FLOG - FLOAT(IEXP)
              FRACT  = 10 ** FRACT
              WRITE(2,310) ICOUNT, FRACT, IEXP
   310        FORMAT(I6,4X,F7.2,' E ',I5)
   320    CONTINUE
C   ***********************************
C   *    FINAL MESSAGE                *
C   ***********************************
          WRITE(2,410)
   410    FORMAT(///' END OF OUTPUT')
          WRITE(1,420)
   420    FORMAT(/' END OF PROGRAM')
          STOP
          END
```

PRINTED OUTPUT

The following printed output results from the test run:

FACTORIAL TABLE

VALUE	FACTORIAL
100	9.33 E 157
101	9.43 E 159

102	9.62 E	161
103	9.90 E	163
104	1.03 E	166
105	1.08 E	168
106	1.15 E	170
107	1.23 E	172
108	1.32 E	174
109	1.44 E	176
110	1.59 E	178

END OF OUTPUT

4.3 Trigonometric

TRIG FUNCTIONS

Trig functions include sine, cosine, and arctangent. The argument is an angle in radians. This angle must represent one revolution or less. Converting angles in degrees to angles in radians uses the fact that 360 degrees of the complete circle corresponds to 2pi radians. The equation

$$D = 57.295781 * R$$

converts radians to degrees. The equation

$$R = D / 57.295781$$

converts degress to radians.

SIN, DSIN

SIN is a single precision function returning the sine of the argument angle. The angle is in radians and must be between the value 0 and pi. DSIN is a double precision function returning the sine of the angle in radians.

COS, DCOS

COS is a single precision function returning the cosine of the argument. The argument is an angle in radians falling between 0 and pi. DCOS is the corresponding double precision function.

ATAN, DATAN

ATAN is a single precision function returning the arctangent of the argument in radians. The argument is the tangent computed using the relationship

$$\text{Tangent} = \text{SIN}(X) \ / \ \text{COS}(X)$$

with X the angle in radians. ATAN returns the angle resulting in the tangent. DATAN is the corresponding double precision function.

PROGRAM GENERATING TABLE OF SINES AND COSINES

The following program generates the table of sines and cosines:

```
C    *************************************
C    *      P0403                         *
C    *************************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         GENERATE TRIGONOMETRIC TABLE
C         FOR SINE AND COSINE.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80
C         MODEL III.
C    *************************************
C    *      ORGANIZATION                  *
C    *************************************
C    INITIAL MESSAGE
C    PROBLEM PARAMETERS
C    PROCESS
C    FINAL MESSAGE
C    *************************************
C    *      VARIABLES                     *
C    *************************************
C    ICOUNT    COUNTER FOR VALUES
C    ISTART    STARTING POINT
C    ISTOP     STOPPING POINT
```

```
C    ISTEP      STEP SIZE
C    DEGREE     ANGLE IN DEGREES
C    RADIAN     ANGLE IN RADIANS
C    SINE       SINE OF ANGLE
C    COSINE     COSINE OF ANGLE
C    ****************************************
C    *     INITIAL MESSAGE                  *
C    ****************************************
          WRITE(1,110)
     110    FORMAT(/' PROGRAM P0403'
        2          //' GENERATE TRIGONOMETRIC TABLE'
        3           /' FOR SINE AND COSINE.')
          WRITE(2,120)
     120    FORMAT('1TRIGONOMETRIC TABLE'
        2          //' DEGREES' ,5X, 'RADIANS',
        3            6X, 'SINE' ,4X, 'COSINE')
C    ****************************************
C    *     PROBLEM PARAMETERS               *
C    ****************************************
          ISTART = 0
          ISTOP  = 100
          ISTEP  = 10
C    ****************************************
C    *     PROCESS                          *
C    ****************************************
          DO 320 ICOUNT = ISTART, ISTOP, ISTEP
             DEGREE = ICOUNT
             RADIAN = DEGREE / 57.29578
             SINE   = SIN(RADIAN)
             COSINE = COS(RADIAN)
             WRITE(2,310) DEGREE, RADIAN, SINE, COSINE
     310      FORMAT(2F10.5,2F10.7)
     320   CONTINUE
C    ****************************************
C    *     FINAL MESSAGE                    *
C    ****************************************
          WRITE(2,410)
     410    FORMAT(////' END OF OUTPUT')
          WRITE(1,420)
     420    FORMAT(/' END OF PROGRAM')
          STOP
          END
```

PRINTED OUTPUT

The following printed output results from the test run:

TRIGONOMETRIC TABLE

DEGREES	RADIANS	SINE	COSINE
0.00000	0.00000	0.0000000	1.0000001
10.00000	.17453	.1736482	.9848078
20.00000	.34907	.3420202	.9396927
30.00000	.52360	.5000000	.8660254
40.00000	.69813	.6427876	.7660445
50.00000	.87266	.7660445	.6427876
60.00000	1.04720	.8660254	.4999999
70.00000	1.22173	.9396927	.3420202
80.00000	1.39626	.9848078	.1736483
90.00000	1.57080	1.0000001	.0000000
100.00000	1.74533	.9848078	-.1736483

END OF OUTPUT

4.4   Function Library

ADDITIONAL BUILT-IN FUNCTIONS

The FORTRAN compiler includes many more built-in functions. The list of functions tends to change somewhat with each release of the language. The language reference manual will list these functions. Review this document for those functions currently implemented.

LIBRARY OF FUNCTIONS

The following functions are among the most useful:

IFIX	Convert a real number to an integer
INT	Truncate real argument giving integer
AINT	Truncate real argument giving real
PEEK	Peek at integer memory address
RAN	Random number generator
AMOD	Real remainder from dividing real divisor (second argument) into real dividend (first argument)

LOCAL INSTALLATION

Each local installation has its own needs. It should develop one or more libraries of functions. These are in addition to the built-in functions available with the compiler. These become powerful tools for the development of programs. The chapter on subroutines shows how this is done.

COMMENTS ON PROGRAM DESIGN

Functions demonstate some important principles concerning the design of computer programs. Each function has only one result. This is an important concept of computer science. The program should be organized into sections. Each section should perform one task. It is not wise to jam too much work into one section. Multiple tasks require multiple sections.

The programmer is not concerned with how the function works. The only concern is that it work. The function may be a complex program with nested loops and many intermediate variables. This detail is not important from the standpoint of the program that uses the function. The temporary variables are local to the function, and are kept completely separate from the main program. This ability to define local variables and complex procedures is an important factor in the design of complex, yet readable, programs.

4.5   Exercises

1. Check the accuracy of the SIN and COS functions. The following relationship should hold:

$$SIN(X)**2 + COS(X)**2 = 1.$$

2. Check the accuracy of the EXP and ALOG functions. Compute a set of natural logarithms and use the EXP function to return a value that should equal that of the original number.

3. Check the accuracy of the square root function by squaring the result and comparing the square of the square root with the original value.

4. The random number function is among the most interesting to use. Use it to generate a set of random numbers.

5. Use the random number function to simulate tossing a coin 1,000 times and counting the number of heads. If the numbers are random in the interval 0 to 1, let a value in the range 0–.5 correspond to a head and a value in the range .5–1 correspond to a tail.

6. The amount $2,500 is deposited into a savings account which earns 7.75 percent interest compounded continuously. Compute the future accumulated value at the end of 3.75 years. This requires the EXP function.

7. The barrel of an artillery piece is at an elevation angle of 22 degrees with respect to the ground. The shell has a muzzle velocity of 1,250 feet per second when fired. Estimate the distance in feet to the point of impact, and the height in feet for the highest point of trajectory for the shell. Disregard wind resistance and factors other than the pull of gravity. Use the trig functions to determine the horizontal and vertical velocities and the laws of motion to determine the desired results.

5 Sequential files

OVERVIEW Most computer files are sequential. Sequential file processing is efficient since processing speeds are faster than random access processing if the file is processed from beginning to end. Sequential files make efficient use of external storage. There is little or no wasted space. This chapter covers the creation and processing of sequential files.

☐

5.1   Writing a Sequential File

SEQUENTIAL FILES

A sequential file stores data sequentially. Processing takes place sequentially from beginning to end. Sequential processing is usually faster than random access processing. Only if processing involves a small fraction of the file is random access faster than sequential access. Sequential files utilize disk storage space efficiently. This chapter discusses sequential files; a later chapter covers random access files.

BLOCKING

The Radio Shack TRSDOS operating system for the Model III allocates disk storage space on the basis of granules. One granule contains 768 bytes consisting of three sectors of 256 bytes each. The Model III diskette contains 40 tracks of 6 granules each. The Model I and Model II employ different granule sizes. The granule

is the smallest unit of disk space for allocation purposes. The sector size of 256 bytes is relatively standard across computer models.

The operating system allocates space for a file in terms of extents. An extent contains one or more physically contiguous granules. A file contains one or more extents. Sequential file processing proceeds, record by record, within a granule and, if necessary, granule by granule within an extent. The system automatically links extents together during processing.

The standard physical record size is the 256 byte sector. The system efficiently processes any logical record size, automatically blocking and deblocking file accesses. Blocking involves packing multiple logical records per sector. Deblocking involves unpacking logical records from sectors. If necessary, the system will span a record across sector boundaries.

DISK PERFORMANCE

The typical five-inch floppy disk revolves at the rate of 300 RPM (revolutions per minute). This gives five revolutions per second or .2 second per revolution, or 200 milliseconds. This is called the latency of the disk drive, i.e., the latency is the time required to make one revolution.

Each disk drive contains a moving head that can be positioned to read one of the tracks of data on the diskette surface. Positioning the head takes time. Track-to-track access varies from three to 40 milliseconds, depending on the disk drive and the software controlling it. Additional time is required for the head to settle down and begin reading data reliably.

From these measures, it is evident that the system cannot make more than three or four disk accesses per second. The most efficient file access methods are those that allow the system to read and write large amounts of data per access. This is the reason why sequential file processing is usually more efficient than random file processing.

The most efficient sequential files are those consisting of one extent. All granules are contiguous. Processing proceeds sequentially from sector to sector within a granule, from granule to granule within the track, and from track to track within the extent.

DATA THROUGHPUT

Each track contains 4,608 bytes for the Radio Shack Model III.
In theory the disk will transfer data at the rate of 23,040 bytes
per second. In practice, the throughput is much less than the
maximum. Most systems will not load the entire track at one time;
they will load a granule-sized block with one access. This permits
a data throughput of 3,840 bytes per second for sequential files.
The system must wait one full revolution before accessing the
next block.

Consider a file containing records of 64 bytes each. The disk
drive can average four accesses per second. Random access tech-
niques limit the data throughput to a maximum of 256 bytes
per second. Sequential access techniques will allow a data through-
put of up to 3,840 bytes per second.

The efficiency of disk I/O (input/output) is highly dependent
on the operating system. TRSDOS is the operating system de-
veloped by Radio Shack for their systems. The larger the block
size employed by the operating system for sequential files, the
faster the processing speeds can become. Large block sizes have
a significant disadvantage, however. They require more internal
memory to hold the block for the system. This internal memory
is called a buffer. Each file must have at least one buffer. Keeping
the buffer size within bounds limits the size of the block for
input/output transfers.

NEW TECHNOLOGY

Performance measures for external storage devices will continue
to improve. The concepts of access times and throughput will
remain valid for the new technologies. A few simple calculations
give measures predicting effective performance regardless of the
technology involved.

READ/WRITE COMMANDS

The WRITE command for writing to a disk file is the same as the
one for writing to the line printer. The command

WRITE(6,210) VALUE, RESULT

will write the values of the variables VALUE and RESULT to unit number 6 using the FORMAT statement 210.

The READ command for reading from a disk file is similar. The command

READ(6,320) VAR1, VAR2

reads values from the device specified by unit number 6 into the variables VAR1 and VAR2. The FORMAT statement for the READ command must be consistent with that used by the WRITE command used to create the file.

OPENING AND CLOSING FILES

A file must be opened before it can be accessed by READ or WRITE statements. The command

CALL OPEN(6,'DATA/DAT',20)

is a subroutine call. It calls the subroutine OPEN giving it the parameters 6, 'DATA/DAT', and 20.

The first parameter is the unit number 6. The second parameter is the disk data file name including the extension /DAT. The third parameter specifies the record length in bytes.

The OPEN subroutine assigns the file name to the unit number. The file name can also specify the disk drive number. The command

CALL OPEN(9,'FILE1/DAT:1',60)

assigns the file named FILE1/DAT on drive 1 to the internal FORTRAN unit number 9.

When finished, the file should be closed. The command

ENDFILE 6

closes the file that has been assigned to unit number 6.

CREATING THE SEQUENTIAL FILE

The program of this section generates a sequential file containing a table of values and the squares of those values. The FORMAT statement

310 FORMAT(I5,2F10.4)

creates records of 25 bytes each. The first five bytes contain an integer value. The following 20 bytes contain two fields of 10 bytes each for real values. There are four digits to the right of the decimal point.

  The following program creates the file named DATA/DAT:

```
C    ************************************
C    *    P0501                         *
C    ************************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         GENERATE SEQUENTIAL
C         TEST DATA FILE.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80.
C    ************************************
C    *    ORGANIZATION                  *
C    ************************************
C    INITIAL MESSAGE
C    PROBLEM PARAMETERS
C    PROCESS
C    FINAL MESSAGE
C    ************************************
C    *    VARIABLES                     *
C    ************************************
C    NUMBER     NUMBER OF VALUES
C    ICOUNT     COUNTER
C    STEP       STEP SIZE FOR VALUE
C    VALUE      CURRENT VALUE
```

```
C    SQUARE    SQUARE OF VALUE
C    *****************************************
C    *    INITIAL MESSAGE                     *
C    *****************************************
         WRITE(1,110)
    110  FORMAT(/' PROGRAM P0501'
      2           //' GENERATE SEQUENTIAL'
      3            /' TEST DATA FILE.')
C    *****************************************
C    *    PROBLEM PARAMETERS                  *
C    *****************************************
         NUMBER = 10
         VALUE  = -5.0
         STEP   = 1.0
C    *****************************************
C    *    PROCESS                             *
C    *****************************************
         CALL  OPEN(6,'DATA/DAT',25)
         DO 320  ICOUNT = 1, NUMBER
             VALUE  = VALUE + STEP
             SQUARE = VALUE * VALUE
             WRITE(6,310) ICOUNT, VALUE, SQUARE
    310      FORMAT(I5,2F10.4)
    320  CONTINUE
         ENDFILE 6
C    *****************************************
C    *    FINAL MESSAGE                       *
C    *****************************************
         WRITE(1,410)
    410  FORMAT(/' END OF PROGRAM')
         STOP
         END
```

TEST RUN

The test run sends the following output to the terminal:

PROGRAM P0501

GENERATE SEQUENTIAL
TEST DATA FILE.

END OF PROGRAM

5.2   Reading a Sequential File

INITIAL FILE

Before it can be read, a sequential file must be created. Most text editors including SCRIPSIT and EDIT do not create appropriate FORTRAN sequential files. The best long-term solution is to find a text editor that will create appropriate sequential files. Another solution is to use interactive data entry from the terminal with a FORTRAN program to create and edit specific files.

FORMAT STATEMENTS FOR READ COMMAND

FORMAT statements define how the data is organized within each record. Integer fields are the same for both READ and WRITE commands. There is a difference when using the F for the format specification.

The WRITE statement uses the F10.4 command for a 10-column field with four digits to the right of the decimal point. The READ statement uses the F10.4 specification slightly differently. To save storage space many files are created without decimal points. The decimal point always occupies the same location. The F10.4 specification on input specifies an assumed decimal point four digit positions from the right of the field. Any decimal point within the input field will override the assumed location.

If decimal points within an input field are included and not assumed, the field F10.0 is often used. If the input field contains no decimal point, the number will have no fractional part.

END OF FILE

The READ command has an optional parameter specifying the statement number if an end-of-file condition is detected during the read operation. The command

READ(6,230,END=690) A, B, C

branches to the statement numbered 690 when the end of the input file is detected.

Both the READ and WRITE commands allow a similar option if the operation detects other file errors. The command

READ(6,230,END=690,ERR=275) A, B, C

branches to statement number 275 for any input error condition.

PROGRAM READING SEQUENTIAL FILE

The following program reads the sequential file created by the previous program:

```
C    ****************************************
C    *     P0502                            *
C    ****************************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         READ SEQUENTIAL
C         TEST DATA FILE.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80.
C    ****************************************
C    *     ORGANIZATION                     *
C    ****************************************
C    INITIAL MESSAGE
C    PROCESS
C    FINAL MESSAGE
C    ****************************************
C    *     VARIABLES                        *
C    ****************************************
C    NUMBER    NUMBER OF VALUES
C    IVALUE    INTEGER VALUE
C    VALUE     CURRENT VALUE
C    SQUARE    SQUARE OF VALUE
C    ****************************************
C    *     INITIAL MESSAGE                  *
C    ****************************************
          WRITE(1,110)
```

```
      110    FORMAT(/' PROGRAM P0502'
        2             //' READ SEQUENTIAL'
        3              /' TEST DATA FILE.')
C    *************************************
C    *    PROCESS                        *
C    *************************************
           CALL OPEN(6,'DATA/DAT',25)
      300    READ(6,310,END=330) IVALUE, VALUE, SQUARE
      310    FORMAT(I5,2F10.0)
           WRITE(2,320) IVALUE, VALUE, SQUARE
      320    FORMAT(I5,2F10.4)
           GO TO 300
      330    ENDFILE 6
C    *************************************
C    *    FINAL MESSAGE                  *
C    *************************************
           WRITE(1,410)
      410    FORMAT(/' END OF PROGRAM')
           STOP
           END
```

PRINTED OUTPUT

The following printed output results from the test run:

```
 1   -4.0000    16.0000
 2   -3.0000     9.0000
 3   -2.0000     4.0000
 4   -1.0000     1.0000
 5    0.0000     0.0000
 6    1.0000     1.0000
 7    2.0000     4.0000
 8    3.0000     9.0000
 9    4.0000    16.0000
10    5.0000    25.0000
```

5.3   Interactive Data Entry

BATCH PROCESSING

Batch processing methods involve executing the program from start to finish without human intervention. There may be some

interaction setting up the program and at the end. Except for these, the program runs to completion automatically.

INTERACTIVE PROCESSING

Interactive processing involves man-machine interaction during the running of the program. Interactive programs are usually easier to use than batch programs. Users appreciate seeing the results immediately rather than hours later as is typical with large batch processing systems.

Batch processing methods make the most efficient use of computer resources. Interactive processing consumes a significant fraction of the computer system's resources in terms of internal memory and computer time. The result is often better use of human resources.

Microcomputers are ideal for interactive processing. They are so inexpensive that concern about the inefficiencies of inter-active processing is virtually eliminated. The efficient use of people is more significant than the efficient use of the computer. Micro-soft FORTRAN for microcomputers permits interaction between the user and the computer at run time. This is a powerful tool for creating software that is easy to use.

BINARY FILES

Using FORMAT statements for the creation of the sequential file requires using matched FORMAT statements for reading that file. The FORMAT statements define the external characteristics of the data. They give the size of the field, the data type, and the placement of the decimal point.

The primary use of formatted data is for printing. The data are suitable for listing directly. Each digit requires one byte in the file and one column position for the printer. Internal binary number representations are entirely different. Internal binary numbers cannot be printed without conversion. The FORMAT statement specifies how the conversion is to take place.

A file created by one program for use by another program should be in internal binary form. No conversion is necessary. Number conversion is a time-consuming operation on any computer system. Eliminating this step increases the processing speed of the computer.

The unformatted READ and WRITE commands differ only in the parameters within the parentheses. They do not include a format statement number. The command

WRITE(6) A, B, C

writes three real values to the file in internal form. The command

READ(6) A, B, C

reads three values from an unformatted file.

The number of bytes per record depends on the sizes of the data types. A standard integer requires two bytes. Single precision real variables require four bytes. Double precision real values require eight bytes each.

TEST SCORES

The application of this section and the following one involves scores on two tests for each of several individuals. The test score file will contain an ID number (integer) and two test scores (real) for each individual. Each record contains 10 bytes of unformatted data.

TERMINAL INPUT

Microsoft FORTRAN for the Radio Shack TRS-80 uses unit number 1 for the terminal and unit number 2 for the printer. WRITE commands using unit number 1 will write to the video screen. READ commands using unit number 1 will obtain data from the keyboard.

PROMPT MESSAGES

An identifying message should precede each keyboard entry. The statements

```
      WRITE(1,310)
310   FORMAT(' ID NUMBER ? ')
      READ(1,320) ID
320   FORMAT(I3)
```

show how this is done.

PROGRAM

The following program uses interactive data entry to create the sequential file of test scores:

```
C    ************************************
C    *      P0503                        *
C    ************************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         CREATE SEQUENTIAL
C         TEST SCORE FILE.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80.
C    ************************************
C    *      ORGANIZATION                 *
C    ************************************
C    INITIAL MESSAGE
C    INITIALIZE
C    INPUT
C    OUTPUT
C    FINAL MESSAGE
C    ************************************
C    *      VARIABLES                    *
C    ************************************
```

```
C    ID          ID NUMBER
C    SCORE1      SCORE FOR TEST 1
C    SCORE2      SCORE FOR TEST 2
C    ************************************
C    *    INITIAL MESSAGE              *
C    ************************************
          WRITE(1,110)
    110   FORMAT(/' PROGRAM P0503'
      2            //' CREATE SEQUENTIAL'
      3             /' TEST SCORE FILE.')
C    ************************************
C    *    INITIALIZE                   *
C    ************************************
          CALL  OPEN(6, 'TEST/DAT' ,10)
          WRITE(1,210)
    210   FORMAT(/' FILE NAME:  TEST/DAT')
          WRITE(1,220)
    220   FORMAT(/' USE ID NUMBER OF 0'
      2             /' TO TERMINATE DATA ENTRY')
C    ************************************
C    *    INPUT                        *
C    ************************************
    300   WRITE(1,310)
    310   FORMAT(/' ID NUMBER ? ')
          READ(1,320) ID
    320   FORMAT(I3)
          IF (ID.EQ.0) GO TO 490
          WRITE(1,330)
    330   FORMAT(/' SCORE FOR'
      2            /' TEST 1 ? ')
          READ(1,340) SCORE1
    340   FORMAT(F4.0)
          WRITE(1,350)
    350   FORMAT(' TEST 2 ? ')
          READ(1,340) SCORE2
C    ************************************
C    *    OUTPUT                       *
C    ************************************
          WRITE(6) ID, SCORE1, SCORE2
          GO TO 300
    490   ENDFILE 6
```

```
C      ************************************
C      *     FINAL MESSAGE                 *
C      ************************************
       WRITE(1,510)
510    FORMAT(/' END OF PROGRAM')
       STOP
       END
```

TEST RUN

The following shows some of the interaction used in the creation of the test score file:

PROGRAM P0503

CREATE SEQUENTIAL
TEST SCORE FILE

FILE NAME: TEST/DAT

USE ID NUMBER OF 0
TO TERMINATE DATA ENTRY

ID NUMBER ? 101

SCORE FOR
TEST 1 ? 25.75
TEST 2 ? 26.75

ID NUMBER ? 117

.

.

.

ID NUMBER ? 0

END OF PROGRAM

5.4   Sequential File Processing

SEQUENTIAL PROCESSING

Sequential processing involves accessing the records one by one from the beginning to the end of the file. This is the most efficient method of accessing all of the records.

STATISTICAL PROCESSING

The records can be accessed in any order for many types of statistical analyses. Because of this sequential files are widely used in statistical analysis and in business applications.

FILE MAINTENANCE

Consider a business accounts receivable system. Each customer has a record giving the customer's name, address, account balance, and other information such as credit limit and amount past due. The accounts receivable file contains the collection of customer records.

The accounts receivable system requires normal updating procedures. These consist of adjusting the account balance for new purchases, merchandise returns, and customer payments. The system will also provide procedures for inserting records for new customers, deleting records of inactive customers, and changing the relatively permanent fields such as name and address. These adjustments and changes to the file fall under the general term of file maintenance.

UNIQUE IDENTIFIER

Each customer must be uniquely defined to avoid confusion. The name and address are usually sufficient. Changes in address do not, however, designate a change in customer. ID numbers, called keys, serve to precisely identify customers. These ID numbers must be unique. One customer should not have more than one ID number. Two or more customers should not share the same ID number. File maintenance requires some means of uniquely identifying the records to be changed, inserted, or deleted. File maintenance is one of the more complex aspects of most business systems, but is needed to keep the computer files up-to-date.

REPORT GENERATION

Another aspect of file processing is report generation. This involves reading the file and producing a summary report. Sequen-

tial processing is the most efficient method for report generation
if the program must scan the entire file.

TEST SCORE REPORT

The following program processes the test score file computing
the average test score for each person and the average score for
each test.

```
C    ****************************************
C    *      P0504                           *
C    ****************************************
C    AUTHOR
C          COPYRIGHT 1982
C          BY LAWRENCE MCNITT.
C    PURPOSE
C          PROCESS SEQUENTIAL
C          TEST SCORE FILE.
C     SYSTEM
C          MICROSOFT FORTRAN
C          RADIO SHACK TRS-80.
C    ****************************************
C    *      ORGANIZATION                    *
C    ****************************************
C    INITIAL MESSAGE
C    INITIALIZE
C    PROCESS
C    SUMMARY
C    FINAL MESSAGE
C    ****************************************
C    *      VARIABLES                       *
C    ****************************************
C    NUMBER    NUMBER OF RECORDS
C    XNUM      REAL NUMBER
C    ID        ID NUMBER
C    SCORE1    SCORE FOR TEST 1
C    SCORE2    SCORE FOR TEST 2
C    AVG       AVERAGE TEST SCORE
C    AVG1      AVERAGE FOR TEST 1
C    AVG2      AVERAGE FOR TEST 2
C    TOTAL1    TOTAL FOR TEST 1
C    TOTAL2    TOTAL FOR TEST 2
```

```
C     ***************************************
C     *     INITIAL MESSAGE                 *
C     ***************************************
            WRITE(1,110)
      110   FORMAT(/' PROGRAM P0503'
          2             //' PROCESS SEQUENTIAL'
          3              /' TEST SCORE FILE.')
C     ***************************************
C     *     INITIALIZE                      *
C     ***************************************
            CALL  OPEN(6, 'TEST/DAT' ,10)
            WRITE(2,210)
      210   FORMAT('1 TEST SCORE SUMMARY'
          2             /' ID NUMBER    AVERAGE')
            TOTAL1 = 0.0
            TOTAL2 = 0.0
            NUMBER = 0
C     ***************************************
C     *     PROCESS                         *
C     ***************************************
      300   READ(6,END=400) ID, SCORE1, SCORE2
            NUMBER = NUMBER + 1
            AVG    = (SCORE1 + SCORE2) / 2.0
            TOTAL1 = TOTAL1 + SCORE1
            TOTAL2 = TOTAL2 + SCORE2
            WRITE(2,310) ID, AVG
      310   FORMAT(/7X,I3,F10.2)
            GO TO 300
C     ***************************************
C     *     SUMMARY                         *
C     ***************************************
      400   XNUM = NUMBER
            AVG1 = TOTAL1 / XNUM
            AVG2 = TOTAL2 / XNUM
            WRITE(2,410) AVG1, AVG2
      410   FORMAT(///' AVERAGE FOR'
          2             //' TEST 1 ',F10.4
          3             //' TEST 2 ',F10.4
          4             ///' END OF OUTPUT')
C     ***************************************
C     *     FINAL MESSAGE                   *
C     ***************************************
```

```
        WRITE(1,510)
510     FORMAT(/' END OF PROGRAM' /)
        STOP
        END
```

PRINTED OUTPUT FROM TEST RUN

The following printed output was produced from the test run:

```
 TEST SCORE SUMMARY
ID NUMBER   AVERAGE
        101       26.25
        103       38.00
        117       28.35

AVERAGE FOR
TEST 1    30.0667
TEST 2    31.6667

END OF OUTPUT
```

5.5   Exercises

1. Write a program to generate a file of quiz scores using the following information:

QUIZ SCORES

Student	Quiz 1	Quiz 2	Quiz 3	Quiz 4
104	8	6	9	8
107	7	4	10	8
112	8	7	9	9
115	8	6	8	7
119	6	5	10	9

2. Write a program to read the file of quiz scores printing the total of the scores for each student, the average score for each student, and the average score for each quiz.

3. Write a program placing the following values into a sequential file:

Customer ID number	Account balance	Credit limit
101	78.56	200.00
112	0.00	200.00
114	24.25	100.00
142	175.45	400.00

4. Write a program that computes and prints the total of the account balances for the file created by the previous program.

5. Write one program that generates 1,000 records of 20 bytes each containing the values and square roots of the first 1,000 integers. Use the FORMAT statement

   FORMAT(2F10.4)

   for output. Write a second program to read the file. Use the format statement

   FORMAT(2F10.0)

   for input. What is the minimum length of time needed to read the 1,000 records?

6. Write one program that generates 1,000 records containing the values and their square roots in internal real form for the integers 1, 2, . . ., 1000. Write a second program to read the unformatted data without doing any operation other than the read. What is the minimum length of time needed to read the 1,000 records? (Each record contains two four-byte fields.)

6 Subscripted variables

OVERVIEW A vector contains a set of values. A matrix is a two-dimensional table of values. Subscripts locate particular values within the vector or matrix. Applications include generating a vector of random values and sorting them into order and computing row averages and column averages for a matrix of text scores.

□

6.1 One Dimension

VECTOR

A vector is an ordered set of values. Individual values are referenced by relative position using subscripts. In FORTRAN and many other programming languages subscripts are enclosed in parentheses immediately following the vector name. The term "array" applies to any subscripted variable regardless of the number of dimensions. A vector is an array having one dimension.

NUMBER REPRESENTATION

FORTRAN naming conventions apply to arrays. Integer variables have names beginning with the letters I through N. Real variables have names beginning with other letters. Arrays can be single precision or double precision real values. They can be 16-bit integers or 32-bit extended integers.

STORAGE ALLOCATION

FORTRAN requires that storage space be pre-allocated to arrays. The DIMENSION statement is the usual method for allocating

space. Each variable is named and its maximum dimension specified. The statement

DIMENSION M(50),A(200)

allocates space for the integer array *M* containing up to 50 values and the single precision real array *A* containing up to 200 values.

The array *M* consumes a total of 100 bytes for its 50 16-bit integers. The array *A* requires 800 bytes for its 200 real values of four bytes each.

Storage allocation is also specified in the variable type specification statements. The statement

INTEGER*4 KVALUE(200)

reserves 800 bytes for the 200 32-bit extended integers. The statement

DOUBLE PRECISION X(200)

reserves 1600 bytes for the array of double precision values.

PROGRAM SIZE

Large arrays result in large programs. For those programs containing large arrays only a fraction of the array is generally used. The amount of memory used depends on the amount of data in the problem. The array size is defined to be large enough to meet the requirements of possible problems.

SUBSCRIPTS

A subscript is an integer specifying the relative position within the vector. The subscript may be a numeric literal or a variable. The expression

VALUE(3) = A ** 2

places the square of the variable A into the third position of the vector named VALUE. The expression

SUM(I) = SUM(I) + X

adds the value of the variable X to the Ith location of the array named SUM.

Storage space for all arrays must be reserved. The value of the subscript must not exceed the maximum dimension of the array. Most FORTRAN systems do not check bounds on computed subscripts at run time. The program will not run reliably if the array dimensions are exceeded. It is up to the programmer to control the subscripting process.

READING AND WRITING ARRAYS

Reading and writing arrays is a common operation. The command

READ(6,210) VALUE(I)

reads a value from the file defined as unit number 6 and places that value in the Ith location of the vector called VALUE. The following loop reads in a set of N values:

```
      DO 220 I = 1, N
          READ(6,210) VALUE(I)
210       FORMAT(F10.0)
220   CONTINUE
```

The READ command is executed N times, once each time through the loop.

FORTRAN allows the looping mechanism to be imbedded within the READ command. The command

READ(6,210) (VALUE(I), I = 1, N)

reads the N values with one execution of the READ command.

If the number of values is the same as the dimension size specified, then the command

READ(6,210) VALUE

fills the array VALUE with the required number of values from the file defined as unit 6.

RECORD LENGTH

The FORMAT statement together with the WRITE statement can define the record length for some systems. Other systems require that the record length be the same for all records of the file.
   Placing the READ or WRITE command within a Do-loop results in one value per record. The following sequence of commands does this:

```
      DO 220 I = 1, N
            WRITE(7,210) VALUE(I)
210         FORMAT(F10.4)
220   CONTINUE
```

The following two commands also write one value per record for most systems:

```
      WRITE(7,210) (VALUE(I), I = 1, N)
210   FORMAT(F10.4)
```

The commands

```
      WRITE(7,210) (VALUE(I), I = 1, N)
210   FORMAT(6F10.4)
```

generate records of 60 bytes each containing six fields of 10 bytes each.
   Programs that read data from a data file must use READ statements and FORMAT statements that are consistent with the program that created the file. Some systems will permit the values to spill over from one record position into the next on reads and writes. Other systems require close control by the programmer.

Most FORTRAN systems require that each execution of a READ command access the next record in the file. Each WRITE command puts data into the next record. Control over blocking multiple values per record requires use of the internal Do-loop generating subscripts within the READ or WRITE command itself.

SORTING

Sorting is a common processing task and may be either internal or external. Internal sorting involves arranging the data values into ascending or descending order within a vector. External sorting involves placing the records of a file into ascending or descending order within a file. The order of placement depends on the values of one or more fields of the records. Numerous utility packages accomplish external sorts and should be used when needed. There is little value in writing special external sort programs. Internal sorting is needed occasionally. It may be necessary to include an internal sort routine within the application program.

SORT BY SELECTION AND EXCHANGE

There are more than a dozen algorithms (methods) for internal sorts. For small sets of data the choice of algorithm is not critical. Large sets of data having more than 500 values require care in the choice of algorithm. The most efficient algorithms for large sets of data are complex and difficult to program.

Sorting by selection and exchange is one of the easiest methods to understand. It works well for small sets of data. This method involves successive sweeps through the remaining un-sorted portion of the array. Each sweep locates the smallest re-maining value and exchanges that value with the first element of the unsorted portion. After the exchange it is in its proper position in the sorted portion of the vector. The computer con-tinues sweeping the remaining unsorted portion until the entire array is in ascending order.

RANDOM NUMBERS

The FORTRAN function RAN(X) generates a single precision real value in the range 0-1. The argument controls the generation

process. An argument of 0.0 returns the last random number generated. An argument of 1.0 returns the next random number of the series. An argument of –1.0 returns a random number from a new series.

Proper control of the argument is one of the subjects of computer simulation modeling. Computer simulation makes extensive use of the random number function. Another application is that of generating data files for testing programs. Certain tests may involve measuring program performance with large data files. If the data does not already exist in computer readable form, developing large test files can be tedious. Using the random number generator is a practical method for generating large test files.

The program of this section does not generate a separate data file. It uses the random number generator to generate the values for the vector to be sorted. The program requests the number of values, generates those values, and sorts them into ascending order. The program generates up to 200 elements and sorts them into order.

PROGRAM

The following program generates a vector of values and sorts them into ascending order:

```
C    ****************************************
C    *     P0601                            *
C    ****************************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         INTERNAL SORT BY
C         SELECTION AND EXCHANGE
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80.
C    ****************************************
C    *     ORGANIZATION                     *
C    ****************************************
C    INITIAL MESSAGE
```

```
C    GENERATE
C    SORT
C    OUTPUT
C    FINAL MESSAGE
C    ************************************
C    *      VARIABLES                   *
C    ************************************
C    X(200)      VALUES
C    NUMBER      NUMBER OF VALUES
C    BEST        BEST VALUE FOUND SO FAR
C    LBEST       LOCATION OF BEST VALUE SO FAR
C    LOC         CURRENT LOCATION FOR EXCHANGE
C    LTRY        LOCATION FOR NEXT TRY
C    LSTOP       LAST LOCATION FOR EXCHANGE
C    LSTART      STARTING LOCATION FOR SEARCH
C    TEMP        TEMPORARY VALUE FOR EXCHANGE
C    ************************************
C    *      INITIAL MESSAGE             *
C    ************************************
         DIMENSION X(200)
         WRITE(1,110)
  110    FORMAT(/' PROGRAM P0601'
     2             //' INTERNAL SORT USING'
     3                /' SELECTION AND EXCHANGE.')
C    ************************************
C    *      GENERATE                    *
C    ************************************
         WRITE(1,210)
  210    FORMAT(/' NUMBER OF VALUES TO GENERATE ? ')
         READ(1,220) NUMBER
  220    FORMAT(I3)
         DO 230 LOC = 1, NUMBER
            X(LOC) = RAN(1.0)
  230    CONTINUE
C    ************************************
C    *      SORT                        *
C    ************************************
         WRITE (1,310)
  310    FORMAT(/' START OF SORT')
         LSTOP = NUMBER - 1
         DO 330 LOC = 1, LSTOP
            BEST    = X(LOC)
```

```
            LBEST   = LOC
            LSTART = LOC + 1
            DO 320 LTRY = LSTART, NUMBER
                IF (X(LTRY).GE.BEST) GO TO 320
                LBEST = LTRY
                BEST = X(LTRY)
  320       CONTINUE
            TEMP      = X(LOC)
            X(LOC)    = BEST
            X(LBEST)  = TEMP
  330   CONTINUE
C   ************************************
C   *    OUTPUT                        *
C   ************************************
        WRITE(1,410)
  410   FORMAT(/' RANDOM VALUES IN ASCENDING ORDER' /)
        LSTOP = 0
  420   LSTART = LSTOP + 1
        LSTOP  = LSTART + 5
        IF (LSTOP.GT.NUMBER) LSTOP = NUMBER
        WRITE(1,430) (X(LOC), LOC=LSTART,LSTOP)
  430   FORMAT(6F10.6)
        IF (LSTOP.LT.NUMBER) GO TO 420
C   ************************************
C   *    FINAL MESSAGE                 *
C   ************************************
        WRITE(1,510)
  510   FORMAT(/' END OF PROGRAM')
        STOP
        END
```

TEST RUN

The following test run demonstrates the program.

PROGRAM P0601

INTERNAL SORT USING
SELECTION AND EXCHANGE.

NUMBER OF VALUES TO GENERATE ? 200

RANDOM VALUES IN ASCENDING ORDER
    .005426   .010518   . . .

6.2   Two Dimensions

MATRICES

A matrix is a two-dimensional table of values. Each element is identified by its relative row and column location. The subscripts are enclosed in parentheses and separated by commas. The command

X(I,J) = 0.0

places the value zero into the position defined as the Ith row and Jth column of the matrix X.

STORAGE ALLOCATION

As with vectors, the DIMENSION statement is the primary statement allocating storage space for matrices. The statement

DIMENSION X(200,20),B(50)

reserves 16,000 bytes for the matrix X. Matrix X contains a maximum of 200 rows and 20 columns. There is space for 4,000 real values of four bytes each. Large arrays require correspondingly large internal storage.

The type specification statements can also reserve space for matrices. The statement

INTEGER*4 NUM(100,10)

specifies that the matrix NUM contains space for 100 rows and 10 columns. Each element is a 32-bit extended integer. The statement

DOUBLE PRECISION TABLE(10,10)

allocates 800 bytes for the table of double precision real values containing 10 rows and 10 columns.

NESTED DO-LOOPS

Operations on matrices usually require nested Do-loops. The Do-loops generate the subscripts for accessing the table. The outer

loop controls the subscript for one of the dimensions. The inner loop controls the subscript for the other dimension.

TEST SCORE ANALYSIS

A teacher needs to perform a test score analysis. The analysis includes computing the total score and the average score for each student. It also involves computing the average score for each test. A nested Do-loop accomplishes the data entry. Each student corresponds to a row of the matrix of test scores, and each test corresponds to a column of the matrix. Each student has a total test score and an average test score. The outer loop controls the row (student) and the inner loop sums the test scores for the student. The summary for each test consists of an average test score. The outer loop controls the test. The inner loop sums the scores for that test over all students.

PROGRAM

The following program uses interactive data entry for inputing the test scores and prints the test summary for each student and each test:

```
C     ************************************
C     *     P0602                        *
C     ************************************
C     AUTHOR
C          COPYRIGHT 1982
C          BY LAWRENCE MCNITT.
C     PURPOSE
C          TEST SCORE ANALYSIS.
C     SYSTEM
C          MICROSOFT FORTRAN
C          RADIO SHACK TRS-80.
C     ************************************
C     *     ORGANIZATION                 *
C     ************************************
C     INITIAL MESSAGE
C     INPUT
C     STUDENT SUMMARY
```

```
C    TEST SUMMARY
C    FINAL MESSAGE
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *     VARIABLES                                        *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    T(40,5)     TEST SCORES
C    NROWS       NUMBER OF STUDENTS
C    NCOLS       NUMBER OF TESTS
C    I           ROW SUBSCRIPT
C    J           COLUMN SUBSCRIPT
C    SUM         SUM OF THE TEST SCORES
C    AVG         AVERAGE TEST SCORE
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *     INITIAL MESSAGE                                  *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          DIMENSION T(40,5)
          WRITE(1,110)
    110   FORMAT(/' PROGRAM P0602'
       2            //' TEST SCORE ANALYSIS'
       3            /' GIVING SUMMARY FOR EACH STUDENT'
       4            /' AND FOR EACH EXAM.')
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *     INPUT                                            *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          WRITE(1,210)
    210   FORMAT(/' NUMBER OF STUDENTS ? ')
          READ(1,220) NROWS
    220   FORMAT(I2)
          WRITE(1,230)
    230   FORMAT(' NUMBER OF TESTS    ? ')
          READ(1,240) NCOLS
    240   FORMAT(I1)
          WRITE(1,250)
    250   FORMAT(/' SCORE FOR')
          DO 290 I = 1, NROWS
             WRITE(1,260) I
    260      FORMAT(/' STUDENT ',I2)
             DO 280 J = 1, NCOLS
                WRITE(1,270) J
    270         FORMAT(' TEST ',I1,' ? ')
                READ(1,275) T(I,J)
```

```
   275             FORMAT(F3.0)
   280         CONTINUE
   290     CONTINUE
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *    STUDENT SUMMARY                         *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         WRITE(2,310)
   310   FORMAT('1TEST SCORE SUMMARY'
      2         //3X, 'STUDENT' ,5X, 'TOTAL' ,3X, 'AVERAGE')
         DO 340 I = 1, NROWS
             SUM = 0.0
             DO 320 J = 1, NCOLS
                 SUM = SUM + T(I,J)
   320       CONTINUE
             AVG = SUM / NCOLS
             WRITE(2,330) I, SUM, AVG
   330       FORMAT(8X,I2,5X,F5.0,F10.5)
   340   CONTINUE
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *    TEST SUMMARY                            *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         WRITE(2,410)
   410   FORMAT(///6X, 'TEST' ,3X, 'AVERAGE')
         DO 440 J = 1, NCOLS
             SUM = 0.0
             DO 420 I = 1, NROWS
                 SUM = SUM + T(I,J)
   420       CONTINUE
             AVG = SUM / NROWS
             WRITE(2,430) J, AVG
   430       FORMAT(9X,I1,F10.5)
   440   CONTINUE
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *    FINAL MESSAGE                           *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         WRITE(2,510)
   510   FORMAT(///' END OF OUTPUT')
         WRITE(1,520)
   520   FORMAT(/' END OF PROGRAM')
         STOP
         END
```

TEST RUN

The following is a test run for the test score analysis program:

PROGRAM P0602

TEST SCORE ANALYSIS
GIVING SUMMARY FOR EACH STUDENT
AND FOR EACH EXAM.

NUMBER OF STUDENTS ? 5
NUMBER OF TEST   ? 3

SCORE FOR

STUDENT 1

TEST 1 ? 95

TEST 2 ? 91

TEST 3 ? 92

STUDENT 2

TEST 1 ? 78

TEST 2 ? 74

TEST 3 ? 71

PRINTED OUTPUT

The following is the printed output generated by the program:

TEST SCORE SUMMARY

STUDENT	TOTAL	AVERAGE
1	278.	92.66666
2	223.	74.33334
3	189.	63.00000
4	250.	83.33334
5	198.	66.00000

TEST	AVERAGE
1	76.20000
2	75.80000
3	75.60000

END OF OUTPUT

6.3   Labeled Data File

FILE MAINTENANCE

File maintenance includes the operations of inserting and deleting records and changing values for existing records. There needs to be provision for creating the initial data file and maintaining it. File maintenance is an important part of any application requiring data files.

UNLABELED FILES

Data files are usually unlabeled. Programs create the data file and read the data file for processing. The programs that process one file must be rewritten to access other files having different data elements. The values are not meaningful when printed unless the report program adds appropriate labels to the output.

GENERAL PURPOSE PROGRAMS

A general purpose file maintenance system is useful for many applications. It can handle the file maintenance duties for many diverse files. It reduces the need for special purpose file maintenance programs.

The method described in this section and the following one is a simple solution. The data file includes structural information giving the file size and the variable names. The program uses the variable names to assist the user in updating the proper values. The file size parameters permit the program to maintain many diverse files.

DATA TYPES

The most complex systems of this nature allow the definition and use of all possible number representations. Such programs will be very large and complex. The program of this section uses only single precision real variables to keep the programming simple. It illustrates the concept of a general purpose file maintenance program.

UNFORMATTED FILES

The files are unformatted to eliminate the burden of formatting the data during input and output. This also provides a consistent word length. Each variable requires four bytes in binary form. If this were not the case the field lengths would vary from one to 16 bytes. The formatting information would then need to be included for each variable.

PROCESS DATA IN MEMORY

Processing large files is on a record-by-record basis. Very few records are in memory at one time. Most of the records are on disk. The program of this section uses an in-memory file maintenance scheme. Loading the entire file into memory simplifies the updating of sequential files. Many text editors use this method. The user saves the updated version after changes are made. The updated version may be under the old name or under a new name.

The method will gain in popularity with the continuing decline in internal memory costs. Small computer systems will provide efficient, high-speed sequential file load and save operations. Updates to the file will take place in memory. When finished, the file will be saved.

CREATING THE INITIAL LABELED DATA FILE

The program that creates the initial labeled data file establishes the number of values per record and generates the initial records. The generalized file maintenance programs of this chapter allow variable labels from one to 12 characters each.

ALPHABETIC FORMATTED INPUT AND OUTPUT

The numeric integer format expression uses the *I* formatting symbol. The standard real format using the *F* formatting symbol has a fixed decimal point location. The scientific notation real format symbol is *E*. The symbol *A* designates alphabetic symbols. The expression A3 designates an alphabetic variable of three character positions. The expression 3A4 designates three alphabetic variables of four bytes each.

Single precision real variables contain four bytes. Standard practice is to use real variables to store alphabetic information. The size is consistent among most brands of computers. It is also possible to store alphabetic data in standard 16-bit integer variables. Such variables cannot contain more than two alphabetic characters.

FILE ORGANIZATION

The labeled file contains data in records of four bytes each. The first four-byte record contains two 16-bit integers in internal binary form. The first integer gives the number of records, and the second gives the number of variables per record. The label for each variable contains 12 bytes. These are contained in three words of four bytes each. They are written from the real variables one word at a time. The data follows the labels and are in internal single precision real form.

PROGRAM

The following program creates the initial labeled data file using interactive data entry:

```
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *    P0603                                                    *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    AUTHOR
C        COPYRIGHT 1982
C        BY LAWRENCE MCNITT.
C    PURPOSE
C        GENERATE LABELED FILE.
C    SYSTEM
C        MICROSOFT BASIC
C        RADIO SHACK TRS-80.
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *    PROGRAM ORGANIZATION                                     *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    INITIAL MESSAGE
C    GET FILE SPECIFICATIONS
C    GET VARIABLE NAMES
C    GET DATA
```

```
C    WRITE FILE
C    FINAL MESSAGE
C    ************************************
C    *    FILE ORGANIZATION                  *
C    ************************************
C    NUMBER OF OBSERVATIONS  (2 BYTES,  INTEGER)
C    NUMBER OF VARIABLES  (2 BYTES,  INTEGER)
C    LABEL FOR VARIABLE 1 (12 BYTES, ALPHANUMERIC=REAL)
C    LABEL FOR VARIABLE 2 (12 BYTES, ALPHANUMERIC=REAL)
C    ETC.
C    VALUE FOR OBS 1,  VAR 1  (4 BYTES,  REAL)
C    VALUE FOR OBS 1,  VAR 2  (4 BYTES,  REAL)
C    ETC.
C    VALUE FOR OBS 2,  VAR 1  (4 BYTES,  REAL)
C    VALUE FOR OBS 2,  VAR 2  (4 BYTES,  REAL)
C    ETC.
C    ************************************
C    *    VARIABLES                          *
C    ************************************
C    X(200,20)      DATA MATRIX OF VALUES
C    VNAMES(20,4)   VARIABLE NAMES
C    FNAME(4)       FILE NAME
C    NOBS           NUMBER OF OBSERVATIONS
C    NVARS          NUMBER OF VARIABLES
C    I              ROW SUBSCRIPT
C    J              COLUMN SUBSCRIPT
C    K              LABEL SUBSCRIPT
C    ************************************
C    *    INITIAL MESSAGE                     *
C    ************************************
         DIMENSION X(200,20),  FNAME(4),  VNAMES(20,3)
         WRITE(1,110)
   110   FORMAT(/' PROGRAM P0603'
      2           //' CREATE INITIAL LABELED'
      3            /' DATA FILE FOR STATISTICAL'
      4            /' ANALYSIS.')
C    ************************************
C    *    GET FILE SPECIFICATIONS             *
C    ************************************
         WRITE(1,210)
   210   FORMAT(/' NAME FOR FILE              ? ')
         READ(1,220) FNAME
```

```
      220    FORMAT(4A4)
             WRITE(1,230)
      230    FORMAT(' NUMBER OF OBSERVATIONS ? ')
             READ(1,240) NOBS
      240    FORMAT(I3)
             WRITE(1,250)
      250    FORMAT(' NUMBER OF VARIABLES     ? ')
             READ(1,260) NVARS
      260    FORMAT(I2)
C     ***********************************
C     *    GET VARIABLE NAMES           *
C     ***********************************
             WRITE(1,310)
      310    FORMAT(/' VARIABLE NAMES MAY HAVE'
         2          /' UP TO 12 CHARACTERS'
         3          //' NAME FOR')
             DO 340 J = 1, NVARS
                 WRITE(1,320) J
      320        FORMAT(' VAR ',I2,' ? ')
                 READ(1,330) (VNAMES(J,K),K=1,3)
      330        FORMAT(3A4)
      340    CONTINUE
C     ***********************************
C     *    GET DATA                     *
C     ***********************************
             WRITE(1,410)
      410    FORMAT(/' ENTER DATA VALUE')
             DO 460 I = 1, NOBS
                 WRITE(1,420) I
      420        FORMAT(/' OBSERVATION ',I2/)
                 DO 450 J=1, NVARS
                     WRITE(1,430) (VNAMES(J,K),K=1,3)
      430            FORMAT(1X,3A4,' ? ')
                     READ(1,440) X(I,J)
      440            FORMAT(F10.0)
      450        CONTINUE
      460    CONTINUE
C     ***********************************
C     *    WRITE FILE                   *
C     ***********************************
             CALL OPEN(6,FNAME,4)
             WRITE(6) NOBS, NVARS
```

```
          DO 520 J = 1, NVARS
              DO 510 K = 1, 3
                  WRITE(6) VNAMES(J,K)
  510         CONTINUE
  520     CONTINUE
          DO 540 I = 1, NOBS
              DO 530 J = 1, NVARS
                  WRITE(6) X(I,J)
  530         CONTINUE
  540     CONTINUE
          ENDFILE 6
C   ************************************
C   *    FINAL MESSAGE                 *
C   ************************************
          WRITE(1,610)
  610     FORMAT(/' END OF PROGRAM')
          STOP
          END
```

TEST RUN

The following program illustrates using the program to define and create the initial labeled data file:

PROGRAM P0603

CREATE INITIAL LABELED
DATA FILE FOR STATISTICAL
ANALYSIS.

NAME FOR FILE	? FILE/DAT
NUMBER OF OBSERVATIONS	? 5
NUMBER OF VARIABLES	? 3

VARIABLE NAMES MAY HAVE
UP TO 12 CHARACTERS

NAME FOR

VAR 1 ? PRE-TEST

VAR 2 ? MID-TERM

VAR 3 ? FINAL-EXAM

ENTER DATA VALUE

OBSERVATION 1

PRE-TEST   ? 71

MID-TERM   ? 87

FINAL-EXAM   ? 93

OBSERVATION 2

PRE-TEST   ? 56

.

.

.

COMMENTS

The test run creates an initial file with the file name FILE/DAT.
It contains five records of three variables each. The labels for the
three variables are PRE-TEST, MID-TERM, and FINAL-EXAM.
The first record contains the values 71, 87, 93. The values for the
other records follow.

6.4   In-Memory File Maintenance

SEQUENTIAL FILE MAINTENANCE

Sequential files must be created and read in strict sequential order.
This usually requires copying an old file to a new file, making
changes to those records that need changing. Changes to existing
records must be made in the order that the records appear in
the file.

     A simpler approach is possible with files small enough to be
loaded into internal memory. The entire file is loaded into and
changes are made in internal memory. After all the changes are
made, the file is written back to external storage. This requires
sufficient internal memory to contain the entire file at one time.

     Large sequential files require a more complicated approach.
The records are copied one at a time from the old file to the new
file. Changes are made to records as they are copied. Keeping
the input file, output file, and the update procedure synchronized
makes this approach more complex. The program cannot reverse
itself and make changes to records that have already been copied
to the new sequential file.

MEMORY TECHNOLOGY

Technological advances have been rapid in the computer's internal logic and memory circuits. This has resulted in rapid decreases in memory costs. The cost of internal memory was $1.00 per byte in 1965. By 1983 internal memory cost $.0005 per byte. In-memory file maintenance was not practical in 1965 because only a limited amount of costly internal memory was available. In-memory file maintenance is gaining acceptance with computers that have inexpensive, high-capacity internal memory.

ADVANTAGES

There are several advantages to in-memory file maintenance. The sequential file is loaded and written out at the fastest possible file transfer speed. This may be from two to 100 times the speed of the most efficient random access disk files.

  The data is treated as if it is a random access file while it is in internal memory. The data consists of a large matrix. Each row constitutes a record, and each column is a variable. Processing does not have to proceed record by record sequentially. Updates may be in any order. The program can quickly scan all of the records for particular values or when computing totals and other statistical measures.

  When the changes have been made, the file is written back to disk. It may be written over the old file or created as a new file. The best approach is to write the file back into a new region with a new name. The old file remains as a backup in case of problems.

MENU SELECTION

The update program gives the user several options. These options include adding new records to the file, changing existing records, displaying the contents of existing records, and listing the file to the line printer. The program displays a menu of the options and has the user select the desired option by number.

CASE SELECTION

The computed GO TO statement distributes the program control to the routines handling the options. These routines return con-

trol to the menu selection routine. The term "case selection"
applies to this process. The program chooses one of the cases and
performs the routine for that case. Case selection is an important
concept from the field of computer science.

The routine for terminating the updating gives the user the
option of saving the updated file. If it is to be saved, the program
asks for the new file name and then saves the file information
including labels.

PROGRAM

The following program reads in the sequential labeled file, per-
forms the desired operations on the file, and then writes the up-
dated file back to the disk:

```
C     ****************************************
C     *     P0604                            *
C     ****************************************
C     AUTHOR
C           COPYRIGHT 1982
C           BY LAWRENCE MCNITT.
C     PURPOSE
C           UPDATE CONTENTS
C           OF LABELED FILE.
C     SYSTEM
C           MICROSOFT FORTRAN
C           RADIO SHACK TRS-80.
C     ****************************************
C     *     PROGRAM ORGANIZATION             *
C     ****************************************
C     INITIAL MESSAGE
C     LOAD FILE
C     MENU
C     ADD OBSERVATIONS
C     CHANGE VALUES
C     DISPLAY OBSERVATION
C     PRINT FILE
C     WRITE FILE
C     FINAL MESSAGE
C     ****************************************
C     *     FILE ORGANIZATION                *
C     ****************************************
```

```
C     NUMBER OF OBSERVATIONS (2 BYTES, INTEGER)
C     NUMBER OF VARIABLES (2 BYTES, INTEGER)
C     LABEL FOR VARIABLE 1 (12 BYTES, ALPHANUMERIC)
C     LABEL FOR VARIABLE 2 (12 BYTES, ALPHANUMERIC)
C     ETC.
C     VALUE FOR OBS 1, VAR 1 (4 BYTES, REAL)
C     VALUE FOR OBS 1, VAR 2 (4 BYTES, REAL)
C     ETC.
C     VALUE FOR OBS 2, VAR 1 (4 BYTES, REAL)
C     VALUE FOR OBS 2, VAR 2 (4 BYTES, REAL)
C     ETC.
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     *     VARIABLES                                    *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     X(200,20)      DATA MATRIX OF VALUES
C     VNAMES(20,3)   VARIABLE NAMES
C     FNAME(4)       FILE NAME
C     NOBS       NUMBER OF OBSERVATIONS
C     NVARS      NUMBER OF VARIABLES
C     NADD       NUMBER OF OBSERVATIONS TO ADD
C     I          CURRENT OBSERVATION
C     J          CURRENT VARIABLE
C     K          LABEL INDEX
C     ISTART     STARTING OBSERVATION
C     ISTOP      LAST OBSERVATION
C     IRESP      USER RESPONSE CODE
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     *     INITIAL MESSAGE                              *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          DIMENSION X(200,20),VNAMES(20,3),FNAME(4)
          WRITE(1,110)
      110 FORMAT(/' PROGRAM P0604'
        2           //' UPDATE DISK DATA FILE'
        3            /' CONTAINING LABELED'
        4            /' VARIABLES.')
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     *     LOAD FILE                                    *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
          WRITE(1,210)
      210 FORMAT(/' NAME OF DATA FILE ? ')
          READ(1,220) FNAME
      220 FORMAT(4A4)
```

```
          CALL  OPEN(6,FNAME,4)
          READ(6) NOBS,NVARS
          DO 240 J = 1, NVARS
              DO 230  K  = 1, 3
                  READ(6)  VNAMES(J,K)
     230      CONTINUE
     240   CONTINUE
          DO 260  I = 1, NOBS
              DO 250  J = 1, NVARS
                  READ(6)  X(I,J)
     250      CONTINUE
     260   CONTINUE
          ENDFILE 6
C   ***********************************
C   *   MENU                          *
C   ***********************************
     300   WRITE(1,310)
     310   FORMAT(/' OPTIONS'
     1              /'    1    ADD NEW OBSERVATIONS'
     2              /'    2    CHANGE EXISTING VALUES'
     3              /'    3    DISPLAY OBSERVATION'
     4              /'    4    PRINT CONTENTS OF FILE'
     5              /'    5    TERMINATE PROCESSING'
     6           //' OPTION NUMBER ? ')
          READ(1,320) IRESP
     320   FORMAT(I1)
          GO TO  (1000,2000,3000,4000,5000), IRESP
          WRITE(1,330)
     330   FORMAT(/' INVALID RESPONSE')
          GO TO 300
C   ***********************************
C   *   ADD OBSERVATIONS              *
C   ***********************************
     1000   WRITE(1,1010)
     1010   FORMAT(/' NUMBER OF OBSERVATIONS TO ADD ? ')
          READ(1,1020) NADD
     1020   FORMAT(I3)
          ISTART = NOBS + 1
          ISTOP  = NOBS + NADD
          NOBS   = ISTOP
          DO 1070 I = ISTART, ISTOP
              WRITE(1,1030) I
```

```
1030          FORMAT(/' OBSERVATION ',I3
     2                 //' VALUE FOR' /)
              DO 1060 J = 1, NVARS
                  WRITE(1,1040) (VNAMES(J,K),K=1,3)
1040              FORMAT(1X,3A4,' ? ')
                  READ(1,1050) X(I,J)
1050              FORMAT(F10.0)
1060          CONTINUE
1070      CONTINUE
          GO TO 300
C    ***********************************
C    *    CHANGE VALUES                *
C    ***********************************
2000      WRITE(1,2010)
2010      FORMAT(/' CHANGE EXISTING VALUES'
     2             /' OF OBSERVATION NUMBER ? ')
          READ(1,2020) I
2020      FORMAT(I3)
          WRITE(1,2030)
2030      FORMAT(/' NO. NAME           CURRENT VALUE')
          DO 2050 J = 1, NVARS
              WRITE(1,2040) J,(VNAMES(J,K),K=1,3),X(I,J)
2040          FORMAT(I3,2X,3A4,F12.5)
2050      CONTINUE
          WRITE(1,2060)
2060      FORMAT(/' USE VAR. NO. OF 0 TO TERMINATE
     2             CHANGES' /' TO THIS OBSERVATION')
2100      WRITE(1,2110)
2110      FORMAT(/' NUMBER OF VARIABLE TO CHANGE ? ')
          READ(1,2120) J
2120      FORMAT(I2)
          IF (J.LT.1) GO TO 2200
          WRITE(1,2130) (VNAMES(J,K) ,K=1,3),X(I,J)
2130      FORMAT(1X,3A4,' OLD VALUE ',F12.5)
          WRITE(1,2140)
2140      FORMAT(13X,' NEW VALUE ? ')
          READ(1,2150) X(I,J)
2150      FORMAT(F10.0)
          GO TO 2100
2200      WRITE(1,2210)
2210      FORMAT(/' CHANGE ANOTHER OBSERVATION (Y/N) ? ')
          READ(1,2220) IRESP
```

```
      2220    FORMAT(A1)
              IF  (A.EQ.'Y')  GO TO 2000
              GO TO 300
C     ************************************
C     *     DISPLAY OBSERVATION                   *
C     ************************************
      3000    WRITE(1,3010)
      3010    FORMAT(/' DISPLAY OBSERVATION NUMBER  ? ')
              READ(1,3020)  I
      3020    FORMAT(I3)
              DO 3040 J = 1, NVARS
                  WRITE(1,3030)  (VNAMES(J,K),K=1,3),X(I,J)
      3030          FORMAT(1X,3A4,2X,F12.5)
      3040    CONTINUE
              WRITE(1,3050
      3050    FORMAT(/' DISPLAY ANOTHER OBSERVATION (Y/N) ?')
              READ(1,3060)  IRESP
      3060    FORMAT(A1)
              IF  (IRESP.EQ.'Y')  GO TO 3000
              GO TO 300
C     ************************************
C     *     PRINT FILE                                      *
C     ************************************
      4000    WRITE(1,4010)
      4010    FORMAT(/' LIST ON PRINTER')
              WRITE(2,4020)
      4020    FORMAT('1FILE CONTENTS')
              WRITE(2,4030) NOBS, NVARS
      4030    FORMAT(I5,' OBSERVATIONS'
          2          /I5,' VARIABLES'
          3          //' VAR NO   LABEL')
              DO 4050 J = 1, NVARS
                  WRITE(2,4040)  J,(VNAMES(J,K),K=1,3)
      4040          FORMAT(5X,I2,3X,3A4)
      4050    CONTINUE
              DO 4100 I = 1, NOBS
                  WRITE(2,4060)  I
      4060          FORMAT(/' OBSERVATION ',I3)
                  ISTOP = 0
      4070          ISTART = ISTOP + 1
                  ISTOP   = ISTART + 5
```

```
                IF (ISTOP.GT.NVARS) ISTOP = NVARS
                WRITE(2,4080) (X(I,J) ,J=ISTART,ISTOP)
     4080       FORMAT(1X,6F12.5)
                IF (ISTOP.LT.NVARS) GO TO 4070
     4100   CONTINUE
           WRITE(2,4110)
     4110   FORMAT(///' END OF OUTPUT')
           GO TO 300
C   ***********************************
C   *     WRITE FILE                  *
C   ***********************************
     5000   WRITE(1,5010)
     5010   FORMAT(/' SAVE FILE TO DISK (Y/N) ? ')
           READ(1,5020) IRESP
     5020   FORMAT(A1)
           IF (IRESP.EQ.'N') GO TO 6000
           WRITE(1,5030)
     5030   FORMAT(/' NAME FOR DISK FILE ? ')
           READ(1,5040) FNAME
     5040   FORMAT(4A4)
           CALL OPEN(6,FNAME,4)
           WRITE(6) NOBS, NVARS
           DO 5060 J = 1, NVARS
               DO 5050 K = 1, 3
                   WRITE(6) VNAMES(J,K)
     5050       CONTINUE
     5060   CONTINUE
           DO 5080 I = 1, NOBS
               DO 5070 J = 1, NVARS
                   WRITE(6) X(I,J)
     5070       CONTINUE
     5080   CONTINUE
           ENDFILE 6
C   ***********************************
C   *     FINAL MESSAGE               *
C   ***********************************
     6000   WRITE(1,6010)
     6010   FORMAT(/' END OF PROGRAM')
           STOP
           END
```

TEST RUN

The following test run illustrates the use of the program for in-memory file maintenance:

PROGRAM P0604

UPDATE DISK DATA FILE
CONTAINING LABELED
VARIABLES

NAME OF DATA FILE ? FILE/DAT

OPTIONS
    1   ADD NEW OBSERVATIONS
    2   CHANGE EXISTING VALUES
    3   DISPLAY OBSERVATION
    4   PRINT CONTENTS OF FILE
    5   TERMINATE PROCESSING

OPTION NUMBER ? 3

DISPLAY OBSERVATION NUMBER ? 1

PRE-TEST	71.00000
MID-TERM	87.00000
FINAL-EXAM	93.00000

DISPLAY ANOTHER OBSERVATION (Y/N) ? N

OPTIONS
    1   ADD NEW OBSERVATIONS
    2   CHANGE EXISTING VALUES
    3   DISPLAY OBSERVATION
    4   PRINT CONTENTS OF FILE
    5   TERMINATE PROCESSING

OPTION NUMBER ? 2

CHANGE EXISTING VALUES
OF OBSERVATION NUMBER ? 3

NO.	NAME	CURRENT VALUE
1	PRE-TEST	42.00000
2	MID-TERM	49.00000
3	FINAL-EXAM	63.00000

USE VAR. NO. OF 0 TO TERMINATE CHANGES
TO THIS OBSERVATION

NUMBER OF VARIABLE TO CHANGE ? 2

OLD VALUE   49.00000
NEW VALUE   ? 59.00000

NUMBER OF VARIABLE TO CHANGE ? 0

CHANGE ANOTHER OBSERVATION (Y/N) ? N

OPTIONS
    1   ADD NEW OBSERVATIONS
    2   CHANGE EXISTING VALUES
    3   DISPLAY OBSERVATION
    4   PRINT CONTENTS OF FILE
    5   TERMINATE PROCESSING

OPTION NUMBER ? 4

LIST ON PRINTER

OPTIONS
    1   ADD NEW OBSERVATIONS
    2   CHANGE EXISTING VALUES
    3   DISPLAY OBSERVATION
    4   PRINT CONTENTS OF FILE
    5   TERMINATE PROCESSING

OPTION NUMBER ? 5

SAVE FILE ON DISK (Y/N)? Y

NAME FOR DISK FILE ? FILE1/DAT

END OF PROGRAM

PRINTED OUTPUT

The following printed output resulted from the program run:

FILE CONTENTS
   5 OBSERVATIONS
   3 VARIABLES

VAR NO   LABEL
     1   PRE-TEST
     2   MID-TERM
     3   FINAL-EXAM

OBSERVATION 1
    71.00000   87.00000   93.00000

OBSERVATION 2
56.00000   74.00000   86.00000

.
.
.

6.5   Exercises

1. Write a program that reads the labeled data file and computes the row sums, row averages, and column averages for the test score data illustrated for the general purpose labeled file system.

2. Use the general purpose program to create a file containing the following data together with labels:

Age	Weight	Blood pressure
43	164	121
59	186	146
37	153	114
46	193	125
64	179	152

3. Use the general purpose file maintenance system to create and maintain an accounts receivable master file. The variables should include an ID number, account balance, and credit limit.

4. Use the general purpose file maintenance system to create and maintain an inventory control system master file. The variables should include an inventory item ID number, balance on hand, number of order, reorder point, and order quantity.

5. Write a program that generates 1,000 random numbers and computes the minimum, maximum, and average of those numbers.

6. Write a program that generates 1,000 random numbers and sorts them into order. How long does the sorting operation take?

7 Subroutines

OVERVIEW There are two types of FORTRAN subroutines, function subroutines and called subroutines. Function subroutines are similar to built-in functions except that they are developed by the user. Called subroutines do not return a value as functions do. The calling program and subroutine pass information back and forth through the parameter list.

☐

7.1   Function Subroutines

RESULT OF FUNCTION

The result of the function call is one value. This value may be integer or real. It may be single or double precision. The numeric type is specified either implicitly by the variable name or explicitly. The command

SQROOT = SQRT(VALUE)

places the square root of the contents of VALUE into the variable SQROOT. The built-in function SQRT has one argument.

USER-DEFINED FUNCTIONS

FORTRAN provides a complete facility for creating and employing user-defined functions. Their use is the same as for built-in functions. User-defined functions are created and compiled independently of the program using them. The compiler creates a relocatable object program. A separate step links the function relocatable and the main program relocatable to form a command file. The command file is loaded and executed directly.

135

SPECIAL FORTRAN STATEMENTS

The first statement within the function is the command FUNC-
TION followed by an argument list. The command

FUNCTION DIAG(ALNGTH,WIDTH)

tells the compiler that this is to be the function named DIAG and
that it will use two real arguments. The names ALNGTH and
WIDTH are dummy variables. The names are local to the func-
tion but their values come from the program that uses the func-
tion. The program using the function passes the addresses of the
variables.

The program may include the command

XLEN = DIAG(A,B)

which uses the function DIAG with the variables *A* and *B* of the
calling program. This facility allows the main program to define
its own variable names independently of those defined within
the function. This feature makes functions easily reusable by
many programs.

Functions use the RETURN command rather than the STOP
command. The RETURN command returns control to the calling
program. The END statement is the last line of the function.

RESULTS

The result of the function is placed in a variable using the same
name as the function name. The calling program accesses this
value. The following example illustrates this feature.

The length of the diagonal of a rectangle is the square root of
the sum of the squares of its length and width. The following FOR-
TRAN function illustrates the form of the function subroutine:

```
FUNCTION  DIAG(ALNGTH,WIDTH)
SQR1    =  ALNGTH * ALNGTH
SQR2    =  WIDTH * WIDTH
SUMSQR =  SQR1 + SQR2
DIAG    =  SQRT(SUMSQR)
RETURN
END
```

The result is placed in the variable DIAG having the same name as the function. The variables ALNGTH and WIDTH are dummy variables whose values come from the calling program. The variables SQR1, SQR2, and SUMSQR are local to the function.

COMPOUND INTEREST

Deposits left in savings accounts earn compound interest. The FORTRAN statement

```
ENDBAL = BEGBAL * (1.0 + RATE) ** NYEARS
```

gives the ending balance at the end of NYEARS for a deposit of BEGBAL earning interest at the rate of RATE compounded annually. The statement

```
ENDBAL = BEGBAL * (1.0 + RATE)
```

gives the ending balance at the end of the first year.

PRESENT VALUE

The present value of an investment is the beginning value which will grow to a specified future value in a given period of time. The statement

```
BEGBAL = ENDBAL / (1.0 + RATE) ** NYEARS
```

gives the present value of the amount ENDBAL to be received in NYEARS assuming a discount rate of RATE. The statement

```
BEGBAL = ENDBAL / (1.0 + RATE)
```

gives the present value of the amount ENDBAL to be received in one year.

DISCOUNTED CASH FLOWS

Many investment problems involve an initial cost followed by one or more years of further costs and returns. The costs and

returns are reduced to a series of annual net cash flows. The sum of the present values of these annual cash flows constitutes the present discounted value of the investment.

DISCOUNT RATE

The discount rate represents the rate of return (interest rate equivalent) of alternative investments. A negative present value results if the proposed investment does not provide the profitability of the alternative investments. A positive net present value results if the proposed investment provides a better return than the alternative investments.

INTERNAL RATE OF RETURN

The internal rate of return is that discount rate which results in a discounted net present value of zero. A set of potential investments can be ranked by their respective internal rates of return. The most desirable investments are those having a high rate of return.

Solving for the internal rate of return requires a trial-and-error process. Bisection is an efficient algorithm to use in this search. Starting with two discount rates that bracket the internal rate of return, the method bisects the interval to obtain a new discount rate. The process continues bisecting the interval until the interval containing the rate of return is sufficiently small.

A positive present value results if the discount rate is too low. A negative present value results if the discount rate is too high. Select a new discount rate exactly midway between the two discount rates that bracket the internal rate of return. Inspect the sign of the present value. The new discount rate becomes one of the boundaries for determining the new midpoint.

PROGRAM DESCRIPTION

The program of this section estimates the internal rate of return for a series of net cash flows. The program searches for two discount rates that bracket the desired internal rate of return. The program then continues the search using the method of bisection.

The main program calls the user-defined function PVALUE which computes the present value of the net cash flows for a

specified discount rate. The function is called from several locations within the main program.

SYSTEM COMMANDS

The Microsoft FORTRAN system for the Radio Shack TRS-80 includes a compiler (F80) and a linker (L80). The command

F80 P0701=P0701

compiles the source program contained in P0701/FOR and creates the relocatable for P0701/REL. The command

F80 S0701=S0701

compiles the source program contained in S0701/FOR and creates the relocatable for S0701/REL.
    The command

L80 P0701-N,S0701,P0701-E

creates the command file P0701/CMD using the subroutine relocatable S0701/REL and the main program relocatable P0701/REL. If a program uses several subroutines, the files containing the relocatables are given to the linker. The command

L80 PROG-N,S1,S2,S3,PROG-E

illustrates the format.

PROGRAM COMMAND

The FORTRAN main program can be named using a command similar to that naming the function. The command

PROGRAM PROG1

assigns the name PROG1 to the main program. The default name is MAIN$.

MAIN PROGRAM

The following program estimates the internal rate of return for a series of net cash flows:

```
          PROGRAM P0701
C    *************************************
C    *    P0701                          *
C    *************************************
C    AUTHOR
C        COPYRIGHT 1982
C        BY LAWRENCE MCNITT.
C    PURPOSE
C        INTERNAL RATE OF RETURN
C        FOR A SERIES OF ANNUAL
C        CASH FLOWS.
C    SYSTEM
C        MICROSOFT FORTRAN
C        RADIO SHACK TRS-80.
C    *************************************
C    *    ORGANIZATION                   *
C    *************************************
C    INITIAL MESSAGE
C    PARAMETERS
C    PROCESS
C        FUNCTION PVALUE
C    OUTPUT
C    FINAL MESSAGE
C    *************************************
C    *    VARIABLES                      *
C    *************************************
C    FLOW(50)   ANNUAL CASH FLOW
C    NUMBER     NUMBER OF ANNUAL CASH FLOWS
C    IYEAR      CURRENT YEAR
C    RATEL      LOWER BOUND FOR RATE
C    RATEU      UPPER BOUND FOR RATE
C    RATE       CURRENT RATE FOR PRESENT VALUE
C    PRVAL      PRESENT VALUE USING CURRENT RATE
C    MAX        MAXIMUM NUMBER OF LOOPS
C    ILOOP      CURRENT LOOP
C    IRESP      USER RESPONSE  (Y/N)
```

```
C     ************************************
C     *     INITIAL MESSAGE              *
C     ************************************
          DIMENSION FLOW(50)
          WRITE(1,110)
110       FORMAT(/' PROGRAM P0701'
     2              //' ESTIMATE THE INTERNAL RATE'
     3              /' OF RETURN GIVING A ZERO'
     4              /' NET DISCOUNTED CASH FLOW'
     5              /' FOR A SERIES OF ANNUAL'
     6              /' CASH FLOWS.')
C     ************************************
C     *     PARAMETERS                   *
C     ************************************
200       WRITE(1,210)
210       FORMAT(/' NUMBER OF ANNUAL CASH FLOWS ? ')
          READ(1,220) NUMBER
220       FORMAT(I2)
          WRITE(1,230)
230       FORMAT(/' CASH FLOW FOR' /)
          DO 260 IYEAR = 1, NUMBER
              WRITE(1,240) IYEAR
240           FORMAT(' YEAR ',I2,' ? ')
              READ(1,250) FLOW(IYEAR)
250           FORMAT(F10.0)
260       CONTINUE
C     ************************************
C     *     PROCESS                      *
C     ************************************
          RATEL = 0.0
          PRVAL = PVALUE(RATEL,NUMBER,FLOW)
          IF (PRVAL.GT.0.0) GO TO 320
          WRITE(1,310)
310       FORMAT(/' SUM OF CASH FLOWS LESS THAN ZERO'
     2              /' DISCOUNTING NOT MEANINGFUL')
          GO TO 500
320       RATEU = RATEL + .2
          PRVAL = PVALUE(RATEU,NUMBER,FLOW)
          IF (PRVAL.LT.0.0) GO TO 330
          RATEL = RATEU
          GO TO 320
```

```
      330   MAXL  = 30
            DO 340 ILOOP = 1, MAXL
                RATE  = (RATEL + RATEU) / 2.0
                PRVAL = PVALUE(RATE,NUMBER,FLOW)
                IF (PRVAL.GT.0.0) RATEL = RATE
                IF (PRVAL.LE.0.0) RATEU = RATE
                IF (ABS(PRVAL).LT.1.0) GO TO 400
      340   CONTINUE
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     *    OUTPUT                                     *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      400   RATE  = 100.0 * RATE
            WRITE(1,410) RATE,PRVAL
      410   FORMAT(/' INTERNAL RATE OF RETURN',F10.4
         2          //' NET PRESENT VALUE      ',F12.2)
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     *    FINAL MESSAGE                              *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      500   WRITE(1,510)
      510   FORMAT(/' TRY ANOTHER PROBLEM  (Y/N) ? ')
            READ(1,520) IRESP
      520   FORMAT(A1)
            IF (IRESP.EQ.'Y') GO TO 200
            WRITE(1,530)
      530   FORMAT(/' END OF PROGRAM' /)
            STOP
            END
```

SUBROUTINE PVALUE

The following subroutine computes the present value for a series of net cash flows using a specified discount rate:

```
            FUNCTION PVALUE(RATE,NUMBER,VALUE)
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     *    S0701                                     *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     AUTHOR
C          COPYRIGHT 1982
C          BY LAWRENCE MCNITT.
C     PURPOSE
```

```
C          COMPUTE THE NET DISCOUNTED
C          PRESENT VALUE FOR A SERIES
C          OF ANNUAL CASH FLOWS.
C    SYSTEM
C          MICROSOFT FORTRAN
C          RADIO SHACK TRS-80.
C    *************************************
C    *     VARIABLES                     *
C    *************************************
C    VALUE(1)   ANNUAL CASH FLOWS
C    NUMBER     NUMBER OF CASH FLOWS
C    RATE       DISCOUNT RATE
C    PVALUE     NET DISCOUNTED PRESENT VALUE
C    IYEAR      CURRENT YEAR
C    INDEX      INDEX FOR LOOP
C    *************************************
C    *     SUBROUTINE                    *
C    *************************************
          DIMENSION VALUE(1)
          PVALUE = 0.0
          DO 100 INDEX = 1, NUMBER
              IYEAR  = NUMBER - INDEX + 1
              PVALUE = VALUE(IYEAR) +  PVALUE /
              (1.0 + RATE)
   100    CONTINUE
          RETURN
          END
```

TEST RUN

The following is a test run using the program to estimate the internal rate of return:

PROGRAM P0701

ESTIMATE THE INTERNAL RATE
OF RETURN GIVING A ZERO
NET DISCOUNTED CASH FLOW
FOR A SERIES OF ANNUAL
CASH FLOWS.

NUMBER OF ANNUAL CASH FLOWS ? 3

CASH FLOW FOR

YEAR 1 ? --15000

YEAR 2 ? 10000

YEAR 3 ? 10000

INTERNAL RATE OF RETURN   21.5234
NET PRESENT VALUE   .29

TRY ANOTHER PROBLEM (Y/N) ? N

END OF PROGRAM

7.2   Subprograms

CALLED SUBROUTINES

In a called subroutine there is no explicit result defined as there
is in a function. A function is limited to one result. Called sub-
routines may have several results. Both functions and called sub-
routine have argument lists. Both include the RETURN command
returning control to the calling program.

FORTRAN COMMANDS

The defining statement of the called subroutine contains the
command SUBROUTINE and the name of the subroutine and its
argument list. The statement

SUBROUTINE SUB(N,A,B)

specifies the name SUB for the subroutine and the variables to
be supplied by the calling program.
    The calling program uses the CALL statement to access the
subroutine. The command

CALL SUB(NUMBER,XSIZE,YSIZE)

illustrates a typical call to the subroutine. The variables from the
main routine and the subroutine are linked by position within
the argument lists. Numeric type specifications must be consistent
between the main program and the subroutine.

DATA ANALYSIS

The program of this section computes summary measures for a set of real values. The main program obtains the values from the terminal, uses a subroutine to compute the summary measures, and then displays the measures.

PROGRAM

The following program obtains the data and controls the analysis:

```
        PROGRAM P0702
C   ************************************
C   *    P0702                         *
C   ************************************
C   AUTHOR
C        COPYRIGHT 1982
C        BY LAWRENCE MCNITT.
C   PURPOSE
C        DESCRIPTIVE MEASURES
C        FOR A SET OF DATA.
C   SYSTEM
C        MICROSOFT FORTRAN
C        RADIO SHACK TRS-80.
C   ************************************
C   *    ORGANIZATION                  *
C   ************************************
C   INITIAL MESSAGE
C   DATA ENTRY
C   PROCESS
C        SUBROUTINE STAT  (S0702)
C   OUTPUT
C   FINAL MESSAGE
C   ************************************
C   *    VARIABLES                     *
C   ************************************
C   DATA(200)     DATA VALUES
C   NUMBER     NUMBER OF VALUES
C   INDEX      INDEX
C   XMAX       MAXIMUM VALUE
C   XMIN       MINIMUM VALUE
```

```
C    AVG         AVERAGE
C    IRESP       USER RESPONSE  (Y/N)
C    **************************************
C    *     INITIAL MESSAGE                 *
C    **************************************
          DIMENSION DATA(200)
          WRITE(1,110)
     110  FORMAT(/' PROGRAM P0702'
        2           //' COMPUTE THE AVERAGE,'
        3            /' MINIMUM, AND MAXIMUM'
        4            /' FOR A SET OF DATA.')
C    **************************************
C    *     DATA ENTRY                      *
C    **************************************
     200  WRITE(1,210)
     210  FORMAT(/' NUMBER OF VALUES ? ')
          READ(1,220) NUMBER
     220  FORMAT(I3)
          WRITE(1,230)
     230  FORMAT(' VALUE FOR' /)
          DO 260 INDEX = 1, NUMBER
              WRITE(1,240) INDEX
     240      FORMAT(' OBS ',I3,' ? ')
              READ(1,250) DATA(INDEX)
     250      FORMAT(F10.0)
     260  CONTINUE
C    **************************************
C    *     PROCESS                         *
C    **************************************
          CALL STAT(NUMBER,DATA,XMIN,XMAX,AVG)
C    **************************************
C    *     OUTPUT                          *
C    **************************************
          WRITE(1,410) XMIN, XMAX, AVG
     410  FORMAT(/' MINIMUM',F15.5
        2           //' MAXIMUM',F15.5
        3           //' AVERAGE',F15.5)
C    **************************************
C    *     FINAL MESSAGE                   *
C    **************************************
          WRITE(1,510)
     510  FORMAT(/' TRY ANOTHER PROBLEM (Y/N) ? ')
          READ(1,520) IRESP
```

```
520   FORMAT(A1)
      IF (IRESP.EQ.'Y')  GO TO 200
      WRITE(1,530)
530   FORMAT(/' END OF PROGRAM' /)
      STOP
      END
```

SUBROUTINE STAT

The following subroutine computes the minimum, maximum, and average for a set of data:

```
      SUBROUTINE STAT(N,X,XMIN,XMAX,AVG)
C     ***********************************
C     *    S0702                        *
C     ***********************************
C     AUTHOR
C          COPYRIGHT 1982
C          BY LAWRENCE MCNITT.
C     PURPOSE
C          MINIMUM, MAXIMUM,
C          AND AVERAGE FOR A
C          SET OF DATA.
C     SYSTEM
C          MICROSOFT FORTRAN
C          RADIO SHACK TRS-80.
C     ***********************************
C     *     VARIABLES                   *
C     ***********************************
C     X(1)      DATA VALUES
C     N         NUMBER OF VALUES
C     I         SUBSCRIPT
C     V         CURRENT VALUE
C     XMIN      MINIMUM VALUE
C     XMAX      MAXIMUM VALUE
C     SUM       SUM OF THE VALUES
C     AVG       AVERAGE
C     ***********************************
C     *     SUBROUTINE                  *
C     ***********************************
      DIMENSION X(1)
      XMIN = X(1)
      XMAX = XMIN
```

```
        SUM   = XMIN
        DO 110 I = 2, N
            V    = X(I)
            SUM = SUM + V
            IF (V.LT.XMIN)  XMIN = V
            IF (V.GT.XMAX) XMAX = V
   110  CONTINUE
        AVG = SUM / FLOAT(N)
        RETURN
        END
```

TEST RUN

The following program illustrates the use of the program computing summary measures:

PROGRAM P0702

COMPUTE THE AVERAGE,
MINIMUM, AND MAXIMUM
FOR A SET OF DATA.

NUMBER OF VALUES ? 5

VALUE FOR

OBS 1 ? 12.0

OBS 2 ? 11.0

OBS 3 ? 19.0

OBS 4 ? 14.0

OBS 5 ? 15.0

MINIMUM   11.00000

MAXIMUM   19.00000

AVERAGE   14.20000

TRY ANOTHER PROBLEM (Y/N) ? N

END OF PROGRAM

7.3   Subroutine Libraries

REUSABLE SUBROUTINES

Subroutines are reusable. Once written and debugged, the same subroutine can be linked to many programs. Many organizations

develop libraries of FORTRAN subroutines for use in the development of FORTRAN programs.

MATHEMATICAL SUBROUTINE PACKAGES

The major computer manufacturers such as IBM and UNIVAC provide subroutine libraries containing hundreds of FORTRAN subroutines for mathematical and statistical analysis. The user does not need to reprogram these subroutines. Availability of large subroutine libraries is one of the reasons why FORTRAN retains its popularity.

MAIN PROGRAM

In some cases the main program becomes a sequence of subroutine calls. The main program reads the data and prints the results. The subroutines perform the processing steps. The parameter lists document the information flow from main program to subroutine and from subroutine to subroutine.

BINOMIAL DISTRIBUTION

The binomial distribution gives probabilities for the number of successes r in n trials for which p is the probability of a success on any one trial and $q = 1 - p$ is the probability of a failure. A coin is tossed ten times. The binomial distribution gives the probability of observing three heads in the ten tosses. The probability of a head on any one toss is .5.

PROGRAM

The following program computes exact and cumulative binomial probabilities using the subroutine BINOM defined in the next section:

```
        PROGRAM P0703
C   ***********************************
C   *     P0703                       *
C   ***********************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
```

```
C    PURPOSE
C         PRINT PROBABILITIES FOR
C         BINOMIAL DISTRIBUTION.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80.
C    *************************************
C    *    ORGANIZATION                   *
C    *************************************
C    INITIAL MESSAGE
C    PARAMETERS
C    PROCESS
C         SUBROUTINE BINOM
C    OUTPUT
C    FINAL MESSAGE
C    *************************************
C    *    VARIABLES                      *
C    *************************************
C    BPROB(200)   BINOMIAL PROBABILITIES
C    NUMBER    NUMBER OF TRIALS
C    PROB      PROBABILITY OF A SUCCESS ON ANY ONE TRIAL
C    BIN       BINOMIAL PROBABILITY
C    CUM       CUMULATIVE PROBABILITY
C    NR        NUMBER OF SUCCESSES
C    IRESP     USER RESPONSE  (Y/N)
C    *************************************
C    *    INITIAL MESSAGE                *
C    *************************************
          DIMENSION BPROB(200)
          WRITE(1,110)
110       FORMAT(/' PROGRAM P0703'
     2              //' GENERATE EXACT AND CUMULATIVE'
     3              /' BINOMIAL PROBABILITIES.')
C    *************************************
C    *    PARAMETERS                     *
C    *************************************
200       WRITE(1,210)
210       FORMAT(/' NUMBER OF TRIALS        ? ')
          READ(1,220) NUMBER
220       FORMAT(I3)
          WRITE(1,230)
230       FORMAT(' PROBABILITY OF A SUCCESS ? ')
```

```
        READ(1,240) PROB
  240   FORMAT(F10.0)
C   *********************************
C   *     PROCESS                    *
C   *********************************
        CALL BINOM(NUMBER,PROB,BPROB)
C   *********************************
C   *     OUTPUT                     *
C   *********************************
        WRITE(2,410) NUMBER, PROB
  410   FORMAT('1BINOMIAL DISTRIBUTION'
     2         //' NUMBER OF TRIALS          ',I5
     3          /' PROBABILITY OF A SUCCESS ',F10.6
     4         //'     R      P(X=R)          P(X.LE.R)')
        CUM = 0.0
        DO 430 NR = 0, NUMBER
            BIN  = BPROB(NR+1)
            CUM = CUM + BIN
            WRITE(2,420) NR, BIN, CUM
  420       FORMAT(I5,2F15.7)
  430   CONTINUE
        WRITE(2,440)
  440   FORMAT(///' END OF OUTPUT')
C   *********************************
C   *     FINAL MESSAGE              *
C   *********************************
        WRITE(1,510)
  510   FORMAT(/' TRY ANOTHER PROBLEM (Y/N) ? ')
        READ(1,520) IRESP
  520   FORMAT(A1)
        IF (IRESP.EQ.'Y') GO TO 200
        STOP
        END
```

TEST RUN

The following test run results from the program of this section:

PROGRAM P0703

GENERATE EXACT AND CUMULATIVE
BINOMIAL PROBABILITIES.

NUMBER OF TRIALS   ? 5
PROBABILITY OF A SUCCESS ? .2
TRY ANOTHER PROBLEM (Y/N) ? N
END OF PROGRAM

PRINTED OUTPUT

The following printed output resulted from the test run:

BINOMIAL DISTRIBUTION

NUMBER OF TRIALS	5	
PROBABILITY OF A SUCCESS	.200000	
R	P(X=R)	P(X.LE.R)
0	.3276844	.3276844
1	.4096056	.7372900
2	.2048028	.9420928
3	.0512007	.9932935
4	.0064001	.9996936
5	.0003200	1.0000136

END OF OUTPUT

7.4   Top-Down Design

MODULAR PROGRAMMING

Modular programming involves dividing a program into a set of cooperating modules. Each module performs one task or a closely related group of tasks. The modules may be sections of one large program. This is the approach used in earlier chapters. The modules may consist of independently compiled functions and subroutines. This is the approach used in this section.

Much has been written on how to organize large programs so that they will be easy to read, easy to modify, reliable, and efficient. The recurring theme is that large programs should be hierarchical in nature with a main routine accessing subsidiary routines which access still lower-level routines, etc. The routines should be as independent of each other as possible. Data in the

form of argument lists should provide the interface between routines. Information is passed back and forth through the means of argument lists.

FORTRAN subroutines and modules are a natural vehicle for creating hierarchically organized programs. The main program calls subroutines, and higher-level subroutines call lower-level subroutines. The main program and the higher-level driver subroutines contain calls to lower levels. Most of the actual processing is done at the lowest levels.

MODULE SIZE

The choice of module size differs from organization to organization. Some prefer small modules, others prefer large ones. Small modules are easier to understand and to debug on an individual basis, but there are more of them to fit together to form a program. Large modules are more complex and more difficult to debug, but there are fewer of them to integrate into a working program.

EFFICIENCY

Formal subroutines require a certain amount of overhead in terms of internal memory and compute cycles. The amount of overhead varies from computer model to computer model. It is usually influenced by the number of addressable registers available to the machine language programmer. Overhead used in calling and returning from a subroutine increases for computers having more addressable internal registers. This results from having to save all or part of these registers with each subroutine call and then having to restore their contents after returning from the subroutine.

MONOLITHIC PROGRAMS

Large, monolithic programs perform all tasks within the program without the use of subroutines. Large programs have a large number of variables. Each task has its variables devoted to housekeeping and variables holding temporary values needed during

the course of the computations. Inventing unique names for the large number of variables required becomes a chore.

Large, monolithic programs include numerous branching statements. The conditional IF . . . GO TO statement and the unconditional GO TO statement are the primary means of controlling the flow of the program. The resulting program can be compared to a bowl of spaghetti. The twisted and turning paths of program flow become impossible to understand.

FLOWCHARTS

Flowcharts are a visual tool designed to bring order out of the chaos of unrestrained branching. Even large, complex flowcharts will not bring order to the largest monolithic programs.

DISCIPLINED PROGRAMMING

Discipline and restraint are required in writing any large program. A few simple rules help. The program should consist of a hierarchical structure of modules. The upper levels consist of calls to subroutines. The lowest levels consist of processing statements performing the elementary reading, writing, and computing.

Each module has one entry point and one exit point. This rule alone brings order out of the chaos of random branches. The flowcharts themselves become modular. A separate flowchart describes the logic for each module. Each box used in a higher-level module expands into a separate flowchart for the module identified.

TOP-DOWN DESIGN

Top-down design is the term used for a method of writing modular programs. It involves subdividing a task into subtasks, and subdividing each subtask into more elementary subtasks, until the most elementary subtasks can be programmed completely with simple, easy-to-understand modules.

The resulting program structure is hierarchical in nature. The program easily translates into a main program calling subroutines which can call lower-level subroutines. Each subroutine contains

its own local variables for temporary results and housekeeping. The argument list provides the interface between the calling program and its subroutines.

BINOMIAL DISTRIBUTION

The programs of this section and the previous section compute binomial probabilities. In this sense they are similar. The difference lies in the increased use of independently compiled called subroutines that are for the program of this section.

MAIN PROGRAM

The main program consists of calls to subroutines to perform the following tasks:

1. Display initial message
2. Get parameters describing distribution
3. Compute probabilities
4. Print exact and cumulative probabilities
5. Display final message

Also included are branching statements controlling the flow of control among the subroutine calls.

The following is the main program:

```
          PROGRAM P0704
C     **************************************
C     *    P0704                           *
C     **************************************
C     AUTHOR
C          COPYRIGHT 1982
C          BY LAWRENCE MCNITT.
C     PURPOSE
C          PRINT BINOMIAL
C          PROBABILITY TABLE.
C     SYSTEM
C          MICROSOFT FORTRAN
C          RADIO SHACK TRS-80.
```

```
C     ****************************************
C     *     ORGANIZATION                     *
C     ****************************************
C     MAIN PROGRAM
C          SUBROUTINE INIT
C          SUBROUTINE PARAM
C          SUBROUTINE BINOM
C          SUBROUTINE TABLE
C          SUBROUTINE FINAL
C     ****************************************
C     *     VARIABLES                        *
C     ****************************************
C     BPROB(200)     BINOMIAL PROBABILITIES
C     NUMBER    NUMBER OF TRIALS
C     PROB      PROBABILITY OF A SUCCESS ON ANY ONE TRIAL
C     IRESP     USER RESPONSE
C     ****************************************
C     *     MAIN PROGRAM                      *
C     ****************************************
          DIMENSION BPROB(200)
      100 CALL INIT(IRESP)
          IF (IRESP.EQ.'N') GO TO 110
          CALL PARAM(NUMBER,PROB)
          CALL BINOM(NUMBER,PROB,BPROB)
          CALL TABLE(NUMBER,PROB,BPROB)
          CALL FINAL(IRESP)
          IF (IRESP.EQ.'Y') GO TO 100
      110 STOP
          END
```

SUBROUTINE INIT

The following subroutine displays the initial message for the binomial distribution program:

```
          SUBROUTINE INIT(IRESP)
C     ****************************************
C     *     S0704A                           *
C     ****************************************
C     AUTHOR
C          COPYRIGHT 1982
C          BY LAWRENCE MCNITT.
```

```
C    PURPOSE
C          DISPLAY INITIAL MESSAGE
C          FOR BINOMIAL DISTRIBUTION.
C    SYSTEM
C          MICROSOFT FORTRAN
C          RADIO SHACK TRS-80.
C    ***********************************
C    *      VARIABLES                  *
C    ***********************************
C    IRESP          USER RESPONSE
C    ***********************************
C    *      SUBROUTINE                 *
C    ***********************************
         WRITE(1,110)
    110  FORMAT(/' PROGRAM P0704'
       2            //' PRINT TABLE OF PROBABILITIES'
       3             /' FOR THE BINOMIAL DISTRIBUTION.')
    120  WRITE(1,130)
    130  FORMAT(/' DO YOU WANT TO CONTINUE  (Y/N) ? ')
         READ(1,140) IRESP
    140  FORMAT(A1)
         IF  (IRESP.EQ.'Y'.OR.IRESP.EQ.'N')  GO TO 150
         GO TO 120
    150  RETURN
         END
```

SUBROUTINE PARAM

The following subroutine gets the parameters for the specific binomial distribution of interest:

```
         SUBROUTINE  PARAM(NUMBER,PROB)
C    ***********************************
C    *      S0704B                     *
C    ***********************************
C    AUTHOR
C          COPYRIGHT 1982
C          BY LAWRENCE MCNITT.
C    PURPOSE
C          GET PARAMETERS FOR
C          BINOMIAL DISTRIBUTION.
C    SYSTEM
```

```
C          MICROSOFT FORTRAN
C          RADIO SHACK TRS-80.
C     *************************************
C     *     VARIABLES                     *
C     *************************************
C     NUMBER    NUMBER OF TRIALS
C     PROB      PROBABILITY OF A SUCCESS
C     *************************************
C     *     SUBROUTINE                    *
C     *************************************
          WRITE(1,110)
      110 FORMAT(/' NUMBER OF TRIALS          ? ')
          READ(1,120) NUMBER
      120 FORMAT(I3)
          WRITE(1,130)
      130 FORMAT(' PROBABILITY OF A SUCCESS ? ')
          READ(1,140) PROB
      140 FORMAT(F10.0)
          RETURN
          END
```

SUBROUTINE BINOM

An efficient recursive algorithm generates the binomial probabilities. The binomial distribution gives probabilities for X successes in n trials for which the probability of a success is p and the probability of a failure is $q = 1 - p$ for any one trial. The probability of $X=0$ successes is $q**n$. The recursive formula

$$P(X) = P(X-1)*(n-X+1)*/(X*q)$$

gives the probability of X successes as a function of the probability of $X-1$ successes.

Logarithms reduce the chance of underflow that would otherwise be a problem during the calculations. The expression

$$n * \log(q)$$

gives the logarithm of the binomial probability of $X=0$ successes. After that, the expression

$$\log(P(X)) = \log(X-1) + \log((n-X+1)*p/(X*q))$$

gives the logarithm of the binomial probability for $X=1, 2, \ldots, n$.

The following program generates the binomial probabilities:

```
        SUBROUTINE  BINOM(NUMBER,PROB,BPROB)
C   ************************************
C   *    S0704C                        *
C   ************************************
C   AUTHOR
C       COPYRIGHT 1982
C       BY LAWRENCE MCNITT.
C   PURPOSE
C       GENERATE PROBABILITIES FOR
C       BINOMIAL DISTRIBUTION.
C   SYSTEM
C       MICROSOFT FORTRAN
C       RADIO SHACK TRS-80.
C   ************************************
C   *    VARIABLES                     *
C   ************************************
C   BPROB(1)   BINOMIAL PROBABILITIES
C   NUMBER     NUMBER OF TRIALS
C   XNUM       NUMBER OF TRIALS  (REAL)
C   PROB       PROBABILITY OF A SUCCESS ON ANY ONE TRIAL
C   FAIL       PROBABILITY OF A FAILURE ON ANY ONE TRIAL
C   BLOG       LOG OF BINOMIAL PROBABILITY
C   ALOG       LOGARITHM CURRENT TERM
C   ATERM      CURRENT TERM FOR RECURSIVE FORMULA
C   BIN        BINOMIAL PROBABILITY
C   NR         CURRENT NUMBER OF SUCCESSES
C   XNR        CURRENT NUMBER OF SUCCESSES  (REAL)
C   ************************************
C   *    SUBROUTINE                    *
C   ************************************
        DIMENSION BPROB(1)
        FAIL   = 1.0 - PROB
        XNUM  = NUMBER
        BLOG  = XNUM * ALOG(FAIL)
        DO 110 NR = 0, NUMBER
            IF (NR.EQ.0) GO TO 100
            XNR   = NR
            ATERM = (XNUM-XNR+1.0) * PROB / (XNR*FAIL)
            BLOG  = BLOG + ALOG(ATERM)
```

```
100       BIN    = 0.0
          IF (BLOG.GT.-50.0) BIN = EXP(BLOG)
          BPROB(NR+1) = BIN
110   CONTINUE
      RETURN
      END
```

SUBROUTINE TABLE

The following subroutine prints the exact and cumulative probabilities:

```
      SUBROUTINE  TABLE(NUMBER,PROB,BPROB)
C   *******************************************
C   *    S0704D                                *
C   *******************************************
C   AUTHOR
C       COPYRIGHT 1982
C       BY LAWRENCE MCNITT.
C   PURPOSE
C       PRINT EXACT AND CUMULATIVE
C       BINOMIAL PROBABILITIES.
C   SYSTEM
C       MICROSOFT FORTRAN
C       RADIO SHACK TRS-80
C       MODEL III.
C   *******************************************
C   *     VARIABLES                            *
C   *******************************************
C   BPROB(1)   BINOMIAL PROBABILITIES
C   NUMBER     NUMBER OF TRIALS
C   PROB       PROBABILITY OF A SUCCESS ON ANY ONE TRIAL
C   BIN        BINOMIAL PROBABILITY
C   CUM        CUMULATIVE PROBABILITY
C   NR         CURRENT NUMBER OF SUCCESSES
C   *******************************************
C   *     SUBROUTINE                           *
C   *******************************************
      DIMENSION BPROB(1)
      WRITE(2,110) NUMBER, PROB
110   FORMAT('1BINOMIAL DISTRIBUTION'
     2          //' NUMBER OF TRIALS        ',I5
```

```
     3              /' PROBABILITY OF A SUCCESS',F10.6
     4             //'      R       P(X=R)       P(X.LE.R) ')
          CUM = 0.0
          DO 130 NR = 0, NUMBER
              BIN  = BPROB(NR+1)
              CUM = CUM + BIN
              WRITE(2,120) NR, BIN,  CUM
   120        FORMAT(/I5,2F15.7)
   130    CONTINUE
          WRITE(2,140)
   140    FORMAT(///' END OF OUTPUT')
          RETURN
          END
```

SUBROUTINE FINAL

The following subroutine displays the final message for the binomial distribution program:

```
          SUBROUTINE  FINAL(IRESP)
C     ************************************
C     *     S0704E                       *
C     ************************************
C     AUTHOR
C          COPYRIGHT 1982
C          BY LAWRENCE MCNITT.
C     PURPOSE
C          FINAL MESSAGE FOR
C          BINOMIAL DISTRIBUTION.
C     SYSTEM
C          MICROSOFT FORTRAN
C          RADIO SHACK TRS-80.
C     ************************************
C     *     VARIABLES                    *
C     ************************************
C     IRESP       USER RESPONSE
C     ************************************
C     *     SUBROUTINE                   *
C     ************************************
   100    WRITE(1,110)
   110    FORMAT(/' TRY ANOTHER PROBLEM  (Y/N)  ?  ')
          READ(1,120) IRESP
```

```
120    FORMAT(A1)
       IF (IRESP.EQ.'Y'.OR.IRESP.EQ.'N')  GO TO 130
       GO TO 100
130    RETURN
       END
```

TEST RUN

The following test run illustrates the use of the program of this section:

```
PROGRAM P0704

PRINT TABLE OF PROBABILITIES
FOR THE BINOMIAL DISTRIBUTION.

DO YOU WANT TO CONTINUE  (Y/N) ?  Y

NUMBER OF TRIALS          ? 5

PROBABILITY OF A SUCCESS ? .2

TRY ANOTHER PROBLEM  (Y/N) ? N

END OF PROGRAM
```

PRINTED OUTPUT

The following printed output resulted from the test run:

```
BINOMIAL DISTRIBUTION
NUMBER OF TRIALS           5
PROBABILITY OF A SUCCESS .2
```

R	P(X=R)	P(X.LE.R)
0	.3276844	.3276844
1	.4096056	.7372900
2	.2048028	.9420928
3	.0512007	.9932935
4	.0064001	.9996936
5	.0003200	1.0000136

7.5   Exercises

1. Write a function computing the minimum of a set of data. Write another function computing the maximum of a set of data. Write a third function computing the average of a set of data. Write the main program using these three functions to compute the minimum, maximum, and average for the set of data.

2. Modify the program of exercise 1 to include called subroutines for the other tasks of data entry and output.

3. Write a hierarchically organized, modular program using the random number function to generate a matrix of 20 rows and 40 columns. Compute and print the row sums and averages and the column sums and averages. Use called subroutines.

4. Write a program that generates 1,000 random numbers, sorts them into order, and prints them out in compact form. Use called subroutines.

5. Write a general purpose file maintenance system similar to that discussed in Chapter 6. Make the programs hierarchical in nature using called subroutines.

6. Write a program that generates a loan payment schedule giving the amount of payment applied to the principal, the amount applied to interest, and the remaining balance of the loan. Test using a beginning loan balance of $10,000, monthly payments of $200, and an interest rate of 10.75 percent. Make the program hierarchical using called subroutines.

8 Matrix methods

OVERVIEW A matrix is a table of values. Matrices are very important in the field of mathematics. Common matrix operations include scalar and matrix addition, scalar and matrix multiplication, transposition, and inversion. FORTRAN subroutine libraries include routines for these matrix operations. This chapter describes several subroutines that could be part of a subroutine library and shows how to define and utilize the subroutines.

☐

8.1   Matrix Manipulation

DEFINITION

A matrix is a two-dimensional table of values. Each element is identified according to its row and its column position. The value $X(I,J)$ is the element in the Ith row and the Jth column. A matrix that has N rows and M columns has a total of N times M elements. A matrix of 20 rows and 30 columns has 20 * 30 = 600 elementary values.

     The FORTRAN main program allocates space for the entire matrix. A matrix of 600 single precision real values requires 2400 bytes using four bytes for each value. A matrix of double precision values requires eight bytes per value.

EXAMPLE MATRICES

The following example matrices are used in the following discussion:

$$A = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 1 & 4 \end{matrix}$$

$$B = \begin{matrix} 2 & 5 \\ 3 & 6 \end{matrix}$$

$$C = \begin{matrix} 4 & 1 & 3 \\ 2 & 2 & 1 \end{matrix}$$

SCALAR ADDITION

A scalar is added to each element of the matrix in scalar addition. Adding the scalar 2 to the matrix *A* results in

$$2 + A = \begin{matrix} 3 & 4 & 5 \\ 6 & 7 & 8 \end{matrix}$$

SCALAR MULTIPLICATION

Each element of the matrix is multiplied by a scalar value in scalar matrix multiplication. Multiplying the matrix *A* by the scalar 2 results in

$$2 * A = \begin{matrix} 2 & 4 & 6 \\ 8 & 10 & 12 \end{matrix}$$

TRANSPOSITION

Transposition is the exchanging of columns and rows of a matrix. The first row becomes the first column. The second row becomes the second column, etc. The matrix *B* is the transposition of the matrix *A*.

MATRIX ADDITION

Matrix addition is the element-by-element addition of two matrices having the same dimensions. Both matrices must have the same number of rows and the same number of columns. Forming the sum of the two matrices *A* and *C* results in

$$A + B = \begin{matrix} 5 & 3 & 6 \\ 6 & 7 & 7 \end{matrix}$$

MATRIX MULTIPLICATION

The matrix product of two matrices computes the value for the element in the *I*th row and *J*th column of the resulting matrix

as the sum of the products of the elements in the Ith row of the first matrix and the Jth column of the second matrix. The number of columns of the first matrix must equal the number of rows of the second matrix. The number of rows of the first matrix becomes the number of rows of the product matrix. The number of columns of the second matrix becomes the number of columns of the product matrix. The matrix product $A*C$ becomes

$$A * C = \begin{matrix} 14 & 32 \\ 32 & 77 \end{matrix}$$

IDENTITY MATRIX

A square matrix has the same number of columns as rows. An identity matrix is a square matrix with the value 1.0 down the diagonal and 0.0 everywhere else. The following is an identity matrix:

$$I = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$$

For any square matrix D, the following matrix products hold:

$$D * I = D$$
$$I * D = D$$

INVERSION

The inverse of the square matrix D is that matrix which gives the identity matrix when multiplied by D. If Inv(D) is the inverse of the matrix D, then the relationships

$$D * Inv(D) = I$$

and

$$Inv(D) * D = I$$

are both true.

The inverse does not exist for every matrix. Such matrices are called singular. The determinant of a square matrix is a measure that identifies singularity. A determinant of zero signifies singularity. A determinant that is almost zero gives a warning that singularity is possible. Round-off problems are greatest with matrices having determinants near the value zero.

PROCESSING METHODS

The operations of addition, multiplication, and transposition are straightforward. Inverting a matrix is not. The best-known method is Gaussian elimination which is the method used in this chapter.

Elementary mathematics courses discuss the Gaussian elimination method starting with the original matrix D and the identity matrix of the same size. Row transformations convert the original matrix into an identity matrix. The same row transformations on the initial identity matrix result in the inverse matrix. The original matrix becomes an identity matrix. The original identity matrix becomes the inverse matrix. If internal memory is limited, the method can be adjusted to form the inverse in place within the space containing the original matrix.

8.2　Subroutine Package

SUBROUTINE LIBRARIES

There are many FORTRAN subroutine packages for large computer systems. These include routines for matrix manipulations. These subroutines can be adapted for use by microcomputers. They may also include matrix inversion methods other than Gaussian elimination which reduce round-off errors.

DIMENSIONS

One of the problems with any subroutine library is a consistent dimensioning of the matrices. If the main program uses one set of dimensions and the subroutines use another set, there will be problems. The subroutine library of this chapter assumes that all matrices are defined with a maximum of 10 rows and 10 columns. This is adequate for small problems. Large matrices of widely differing dimensions require special handling.

Many subroutines redefine the matrix as a one-dimensional vector. Each matrix access requires calculation to convert the row and column subscripts to the appropriate location. The main

program defines the matrix as two-dimensional. The subroutine call gives the starting address of the matrix and the matrix dimensions in the parameter list. The subroutine uses that information to calculate the relative position.

SCALAR ADDITION

The following subroutine adds a scalar and a matrix:

```
      SUBROUTINE SADD(SCALAR,NROWS,NCOLS,
      AMAT,RMAT)
C    ************************************
C    *    S08A                          *
C    ************************************
C    AUTHOR
C        COPYRIGHT 1982
C        BY LAWRENCE MCNITT.
C    PURPOSE
C        ADD A SCALAR
C        TO A MATRIX.
C    SYSTEM
C        MICROSOFT FORTRAN
C        RADIO SHACK TRS-80.
C    ************************************
C    *    VARIABLES                     *
C    ************************************
C    AMAT(10,10)   ORIGINAL MATRIX
C    RMAT(10,10)    RESULT MATRIX
C    SCALAR     VALUE OF SCALAR
C    NROWS      NUMBER OF ROWS
C    NCOLS      NUMBER OF COLUMNS
C    I          ROW SUBSCRIPT
C    J          COLUMN SUBSCRIPT
C    ************************************
C    *    SUBROUTINE                    *
C    ************************************
      DIMENSION AMAT(10,10),RMAT(10,10)
      DO 120 I = 1, NROWS
          DO 110 J = 1, NCOLS
              RMAT(I,J) = SCALAR + AMAT(I,J)
```

```
       110        CONTINUE
       120    CONTINUE
              RETURN
              END
```

SCALAR MULTIPLICATION

The following program forms the product of a scalar and a matrix:

```
          SUBROUTINE  SMULT(SCALAR,NROWS,NCOLS,
          AMAT,RMAT)
C    ************************************
C    *    S08B                          *
C    ************************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         MULTIPLY A SCALAR
C         AND A MATRIX.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80.
C    ************************************
C    *    VARIABLES                     *
C    ************************************
C    AMAT(10,10)    ORIGINAL MATRIX
C    RMAT(10,10)    RESULT MATRIX
C    SCALAR    VALUE OF SCALAR
C    NROWS     NUMBER OF ROWS
C    NCOLS     NUMBER OF COLUMNS
C    I              ROW SUBSCRIPT
C    J              COLUMN SUBSCRIPT
C    ************************************
C    *    SUBROUTINE                    *
C    ************************************
          DIMENSION AMAT(10,10),RMAT(10,10)
          DO 120 I = 1, NROWS
              DO 110 J = 1, NCOLS
                  RMAT(I,J) = SCALAR * AMAT(I,J)
```

```
110        CONTINUE
120     CONTINUE
        RETURN
        END
```

TRANSPOSITION

The following subroutine transposes a matrix:

```
        SUBROUTINE  TRAN(NROWS,NCOLS,AMAT,RMAT)
C   ************************************
C   *    S08C                          *
C   ************************************
C   AUTHOR
C        COPYRIGHT 1982
C        BY LAWRENCE MCNITT.
C   PURPOSE
C        MATRIX TRANSPOSITION.
C   SYSTEM
C        MICROSOFT FORTRAN
C        RADIO SHACK TRS-80.
C   ************************************
C   *    VARIABLES                     *
C   ************************************
C   AMAT(10,10)   ORIGINAL MATRIX
C   RMAT(10,10)    RESULT MATRIX
C   NROWS      NUMBER OF ROWS
C   NCOLS      NUMBER OF COLUMNS
C   I          ROW SUBSCRIPT
C   J          COLUMN SUBSCRIPT
C   ************************************
C   *    SUBROUTINE                    *
C   ************************************
        DIMENSION AMAT(10,10),RMAT(10,10)
        DO 120 I = 1, NROWS
            DO 110 J = 1, NCOLS
                RMAT(J,I) = AMAT(I,J)
110         CONTINUE
120     CONTINUE
        RETURN
        END
```

MATRIX ADDITION

The following subroutine performs matrix addition:

```
         SUBROUTINE MADD(NROWS,NCOLS,AMAT,BMAT,RMAT)
C    ****************************************
C    *    S08D                              *
C    ****************************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         ADDITION OF
C         TWO MATRICES.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80.
C    ****************************************
C    *    VARIABLES                         *
C    ****************************************
C    AMAT(10,10)    FIRST MATRIX
C    BMAT(10,10)    SECOND MATRIX
C    RMAT(10,10)    RESULT MATRIX
C    NROWS      NUMBER OF ROWS
C    NCOLS      NUMBER OF COLUMNS
C    I          ROW SUBSCRIPT
C    J          COLUMN SUBSCRIPT
C    ****************************************
C    *    SUBROUTINE                        *
C    ****************************************
         DIMENSION AMAT(10,10),BMAT(10,10),RMAT(10,10)
         DO 120 I = 1, NROWS
             DO 110 J = 1, NCOLS
                 RMAT(I,J) = AMAT(I,J) + BMAT(I,J)
110          CONTINUE
120      CONTINUE
         RETURN
         END
```

MATRIX MULTIPLICATION

The following subroutine performs the matrix multiplication:

```
          SUBROUTINE  MMULT(N1,N2,N3,AMAT,BMAT,RMAT)
C    *************************************
C    *    S08E                           *
C    *************************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         MATRIX MULTIPLICATION.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80.
C    *************************************
C    *    VARIABLES                      *
C    *************************************
C    AMAT(10,10)    FIRST MATRIX
C    BMAT(10,10)    SECOND MATRIX
C    RMAT(10,10)    RESULT MATRIX
C    N1         NUMBER OF ROWS OF FIRST MATRIX
C    N2         NUMBER OF COLUMNS OF FIRST MATRIX
C              AND ROWS OF SECOND MATRIX
C    N3         NUMBER OF COLUMNS OF SECOND MATRIX
C    I          SUBSCRIPT
C    J          SUBSCRIPT
C    K          SUBSCRIPT
C    SUM        SUM OF PRODUCTS
C    *************************************
C    *    SUBROUTINE                     *
C    *************************************
          DIMENSION AMAT(10,10),BMAT(10,10),RMAT(10,10)
          DO 130  I = 1,  N1
             DO 120  J = 1,  N3
                SUM = 0.0
                DO 110  K = 1,  N2
                   SUM = SUM + AMAT(I,K) * BMAT(K,J)
```

```
110        CONTINUE
           RMAT(I,J) = SUM
120        CONTINUE
130   CONTINUE
      RETURN
      END
```

INVERSION

Gaussian elimination is a complex process. For further specifics refer to a textbook on linear algebra or college mathematics. The operations are time-consuming to carry out by hand, but are easily performed by the computer. The operations for Gaussian elimination are done row by row, transforming the original matrix into the inverse matrix.

The following subroutine uses Gaussian elimination to invert a matrix:

```
      SUBROUTINE MINV(NSIZE,AMAT,RMAT,DET)
C    ***********************************
C    *   S08F                          *
C    ***********************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         MATRIX INVERSION.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80.
C    ***********************************
C    *   VARIABLES                     *
C    ***********************************
C    AMAT(10,10)   ORIGINAL MATRIX
C    RMAT(10,10)   RESULT MATRIX
C    LROW(10)      LOCATION VECTOR FOR ROWS
C    ROW(10)       PIVOT ROW
C    COL(10        PIVOT COLUMN
C    DET       DETERMINANT
C    NSIZE     MATRIX DIMENSION
```

```
C    I          SUBSCRIPT
C    J          SUBSCRIPT
C    K          SUBSCRIPT
C    IPIVOT     PIVOT ROW AND COLUMN INDEX
C    LPIVOT     LOCATION OF PIVOT ROW
C    PIVOT      PIVOT ELEMENT
C    IROW       ROW INDEX
C    JROW       ROW INDEX
C    TEMP       TEMPORARY VALUE
C    LTEMP      TEMPORARY VALUE
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *    SUBROUTINE                               *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         DIMENSION AMAT(10,10),RMAT(10,10),
      2              COL(10),ROW(10),LROW(10)
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *    INITIALIZE WORK AREAS                    *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         DO 120 I = 1, NSIZE
             DO 110 J = 1, NSIZE
                 RMAT(I,J) = AMAT(I,J)
110          CONTINUE
120      CONTINUE
         DO 130 I = 1, NSIZE
             LROW(I) = I
130      CONTINUE
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *    GAUSSIAN ELIMINATION                     *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
         DET = 1.0
         DO 190 IPIVOT = 1, NSIZE
C            * * * * * * * * * * * * * * * * * * * * * * * * *
C            *    SELECT PIVOT ROW          *
C            * * * * * * * * * * * * * * * * * * * * * * * * *
             PIVOT = 0.0
             DO 140 I = IPIVOT, NSIZE
                 J     = LROW(I)
                 TEMP  = RMAT(J,IPIVOT)
                 IF (ABS(TEMP).LE.ABS(PIVOT) ) GO TO 140
                 PIVOT = TEMP
                 IROW  = I
```

```
      140         CONTINUE
                  LPIVOT = LROW(IROW)
C                 ****************************
C                 *   SAVE PIVOT COLUMN      *
C                 ****************************
                  DO 145 I = 1, NSIZE
                      COL(I) = RMAT(I,IPIVOT)
                      RMAT(I,IPIVOT) = 0.0
      145         CONTINUE
                  RMAT((LPIVOT,IPIVOT) = 1.0
C                 ****************************
C                 *   CHECK FOR SINGULARITY  *
C                 ****************************
                  DET = DET * PIVOT
                  IF (DET.EQ.0.0) GO TO 999
C                 ****************************
C                 *   TRANSFORM PIVOT ROW    *
C                 ****************************
                  LTEMP = LROW(IPIVOT)
                  LROW(IPIVOT) = LROW(IROW)
                  LROW(IROW) = LTEMP
                  DO 150 J = 1, NSIZE
                      ROW(J) = RMAT(LPIVOT,J) / PIVOT
                      RMAT(LPIVOT,J) = ROW(J)
      150         CONTINUE
C                 ****************************
C                 *   SWEEP MATRIX           *
C                 ****************************
                  DO 170 I = 1, NSIZE
                      IF (I.EQ.LPIVOT) GO TO 170
                      TEMP = COL(I)
                      DO 160 J = 1, NSIZE
                          RMAT(I,J) = RMAT(I,J) - TEMP * ROW(J)
      160             CONTINUE
      170         CONTINUE
      190     CONTINUE
C     ****************************************
C     *   INTERCHANGE ROWS AND COLUMNS       *
C     ****************************************
```

```
        DO 940 I = 1, NSIZE
            IROW = LROW(I)
            IF (IROW.EQ.I) GO TO 940
            DO 910 J = I, NSIZE
                JROW = LROW(J)
                IF (JROW.NE.I) GO TO 910
                LROW(J) = IROW
                LROW(I) = JROW
                GO TO 920
910         CONTINUE
920         DO 925 J = 1, NSIZE
                TEMP = RMAT(IROW,J)
                RMAT(IROW,J) = RMAT(JROW,J)
                RMAT(JROW,J) = TEMP
925         CONTINUE
            DO 930 J = 1, NSIZE
                TEMP = RMAT(J,IROW)
                RMAT(J,IROW) = RMAT(J,JROW)
                RMAT(J,JROW) = TEMP
930         CONTINUE
940     CONTINUE
999     RETURN
        END
```

MATRIX INPUT

The following is a general-purpose read routine for obtaining matrix information from the terminal:

```
        SUBROUTINE MREAD(NROWS,NCOLS,RMAT)
C   ***********************************
C   *    S08G                         *
C   ***********************************
C   AUTHOR
C       COPYRIGHT 1982
C       BY LAWRENCE MCNITT.
C   PURPOSE
C       INTERACTIVE DATA
C       ENTRY FOR MATRIX.
```

```
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80.
C    *****************************************
C    *    VARIABLES                          *
C    *****************************************
C    RMAT(10,10)    RESULT MATRIX
C    NROWS      NUMBER OF ROWS
C    NCOLS      NUMBER OF COLUMNS
C    I              ROW SUBSCRIPT
C    J              COLUMN SUBSCRIPT
C    *****************************************
C    *    SUBROUTINE                         *
C    *****************************************
          DIMENSION RMAT(10,10)
          WRITE(1,110)
     110  FORMAT(/' ENTER VALUES FOR MATRIX'
         2        //' NUMBER OF ROWS      ? ')
          READ(1,120) NROWS
     120  FORMAT(I2)
          WRITE(1,130)
     130  FORMAT(' NUMBER OF COLUMNS ? ')
          READ(1,120) NCOLS
C    *****************************************
C    *    GET MATRIX VALUES                  *
C    *****************************************
          WRITE(1,210)
     210  FORMAT(' VALUE FOR' /)
          DO 260 I = 1, NROWS
               WRITE(1,220) I
     220       FORMAT(' ROW ',I2/)
               DO 250 J = 1, NCOLS
                    WRITE(1,230) J
     230            FORMAT('   COL ',I2,' ? ')
                    READ(1,240) RMAT(I,J)
     240            FORMAT(F10.0)
     250       CONTINUE
     260  CONTINUE
          RETURN
          END
```

MATRIX PRINT SUBROUTINE

The following is a general-purpose routine for printing the contents of a subroutine:

```
          SUBROUTINE  MPRINT(NROWS,NCOLS,RMAT)
C    ************************************
C    *    S08H                          *
C    ************************************
C    AUTHOR
C        COPYRIGHT 1982
C        BY LAWRENCE MCNITT.
C    PURPOSE
C        PRINT CONTENTS
C        OF DATA MATRIX.
C    SYSTEM
C        MICROSOFT FORTRAN
C        RADIO SHACK TRS-80
C        MODEL III.
C    ************************************
C    *    VARIABLES                     *
C    ************************************
C    RMAT(10,10)    DATA MATRIX
C    NROWS      NUMBER OF ROWS
C    NCOLS      NUMBER OF COLUMNS
C    I          ROW SUBSCRIPT
C    J          COLUMN SUBSCRIPT
C    ************************************
C    *    SUBROUTINE                    *
C    ************************************
          DIMENSION RMAT(10,10)
          WRITE(2,110)
110       FORMAT(///' CONTENTS OF MATRIX')
          DO 140 I = 1, NROWS
             WRITE(2,120) I
120          FORMAT(/' ROW ',I2)
             WRITE(2,130) (RMAT(I,J),J=1,NCOLS)
130          FORMAT(5F16.6)
140       CONTINUE
          RETURN
          END
```

MATRIX MOVE

The following subroutine moves a matrix from one location to another:

```
          SUBROUTINE MOVE(NROWS,NCOLS,AMAT,RMAT)
C    ****************************************
C    *    S08I                              *
C    ****************************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         MOVE ONE MATRIX TO
C         ANOTHER LOCATION.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80.
C    ****************************************
C    *    VARIABLES                         *
C    ****************************************
C    AMAT(10,10)    ORIGINAL MATRIX
C    RMAT(10,10)    DESTINATION
C    NROWS     NUMBER OF ROWS
C    NCOLS     NUMBER OF COLUMNS
C    I              ROW SUBSCRIPT
C    J              COLUMN SUBSCRIPT
C    ****************************************
C    *    SUBROUTINE                        *
C    ****************************************
          DIMENSION AMAT(10,10),RMAT(10,10)
          DO 120 I = 1, NROWS
              DO 110 J = 1, NCOLS
                  RMAT(I,J) = AMAT(I,J)
110           CONTINUE
120       CONTINUE
          RETURN
          END
```

8.3   General Purpose Program

MATRIX MANIPULATION

Most programs using the subroutine library perform their own input of the data and the printing of the results. They use only those subroutines needed for the analysis. Top-down design and modular programming techniques encourage the use of subroutines for data entry and printing as well.

GENERALIZED MATRIX PROCESSOR

This section illustrates a general-purpose program for manipulating matrices. It is primitive in nature. Each matrix must contain no more than 10 rows and 10 columns. The program can store no more than four matrices at one time.

The user can enter values for a matrix from the keyboard, control the matrix processing, and display the contents of any of the four matrices. The purpose is to show how to incorporate the subroutines into working programs and to test the operation of the subroutines.

PROGRAM

The following program is a general-purpose program for manipulating matrices:

```
        PROGRAM P0803
C   ***********************************
C   *    P0803                        *
C   ***********************************
C   AUTHOR
C        COPYRIGHT 1982
C        BY LAWRENCE MCNITT.
C   PURPOSE
C        GENERAL-PURPOSE
C        MATRIX PROCESSOR.
C   SYSTEM
C        MICROSOFT FORTRAN
```

```
C          RADIO SHACK TRS-80.
C    ************************************
C    *    ORGANIZATION                   *
C    ************************************
C    INITIAL MESSAGE
C    COMMAND
C    HELP
C    SCALAR ADDITION
C    MATRIX ADDITION
C    SCALAR MULTIPLICATION
C    MATRIX MULTIPLICATION
C    TRANSPOSITION
C    INVERSION
C    MOVE
C    READ
C    PRINT
C    FINAL MESSAGE
C    ************************************
C    *    VARIABLES                      *
C    ************************************
C    A(10,10)   MATRIX 1
C    B(10,10)   MATRIX 2
C    C(10,10)   MATRIX 3
C    D(10,10)   MATRIX 4
C    X(10,10)   TEMPORARY MATRIX
C    Y(10,10)   TEMPORARY MATRIX
C    NROW(4)    VECTOR OF ROW DIMENSIONS
C    NCOLV(4)   VECTOR OF COLUMN DIMENSIONS
C    IRESP       USER COMMAND AND OPERANDS
C    NROWS      NUMBER OF ROWS
C    NCOLS      NUMBER OF COLUMNS
C    NROWS2     NUMBER OF ROWS
C    NCOLS2     NUMBER OF COLUMNS
C    ICODE      CODE NUMBER FOR COMMAND
C    DET        DETERMINANT FROM INVERSION
C    MAT1       INDEX FOR FIRST OPERAND
C    MAT2       INDEX FOR SECOND OPERAND
C    MAT3       INDEX FOR THIRD OPERAND
C    SCALAR     VALUE OF SCALAR
C    ************************************
C    *    INITIAL MESSAGE                *
C    ************************************
```

```
        DIMENSION A(10,10),B(10,10),C(10,10),D(10,10),
    2               X(10,10),Y(10,10),NROWV(4),NCOLV(4)
        WRITE(1,110)
110     FORMAT(/' PROGRAM P0803'
    2              //' GENERAL-PURPOSE MATRIX'
    3               /' MANIPULATIONS.'
    4              //' GIVE COMMAND HELP'
    5               /' TO DISPLAY INSTRUCTIONS.')
C   ************************************
C   *    COMMAND                      *
C   ************************************
200     WRITE(1,210)
210     FORMAT(/' COMMAND ? ')
        READ(1,220) IRESP, ICODE
220     FORMAT(2A2)
        ICODE = 0
        IF (IRESP.EQ'HE') ICODE = 1
        IF (IRESP.EQ.'SA') ICODE = 2
        IF (IRESP.EQ.'MA') ICODE = 3
        IF (IRESP.EQ.'SM') ICODE = 4
        IF (IRESP.EQ.'MM') ICODE = 5
        IF (IRESP.EQ.'TR') ICODE = 6
        IF (IRESP.EQ.'IN') ICODE = 7
        IF (IRESP.EQ.'MO') ICODE = 8
        IF (IRESP.EQ.'RE') ICODE = 9
        IF (IRESP.EQ.'WR') ICODE = 10
        IF (IRESP.EQ.'ST') ICODE = 11
        GO TO (1000,2000,3000,4000,5000,6000,7000,
    2           8000,9000,10000,11000), ICODE
        WRITE(1,230)
230     FORMAT(/' INVALID COMMAND NAME')
        GO TO 200
C   ************************************
C   *    HELP                         *
C   ************************************
1000    WRITE(1,1010)
1010    FORMAT(///' COMMAND      EXPLANATION'
    2           /' HELP     DISPLAY INSTRUCTIONS'
    3           /' SADD     SCALAR ADDITION'
    4           /' MADD     MATRIX ADDITION'
    5           /' SMLT     SCALAR MULTIPLICATION'
    6           /' MMLT     MATRIX MULTIPLICATION')
```

```
         WRITE(1,1015)
  1015   FORMAT(' TRAN      TRANSPOSITION'
     2         /' INVT    INVERSION'
     3         /' MOVE     MOVE MATRIX'
     4         /' READ     READ MATRIX'
     5         /' WRIT     WRITE MATRIX'
     6         /' STOP      TERMINATE PROGRAM'
     7         /'       TYPE THE VALUE 1 TO CONTINUE  ')
         READ(1,1020) RESP
  1020   FORMAT(4A4)
         WRITE(1,1030)
  1030   FORMAT(/' THE FOUR AVAILABLE MATRICES ARE'
     2         /' NUMBERED 1, 2, 3, AND 4.  THE'
     3         /' COMMAND NAME CONTAINS FOUR LETTERS.')
         GO TO 200
C    *****************************************
C    *    SCALAR ADDITION                    *
C    *****************************************
  2000   WRITE(1,2010)
  2010   FORMAT(/' SCALAR ADDITION'
     2            //' MATRIX NUMBER  ? ')
         READ(1,2020) MAT1
  2020   FORMAT(I1)
         IF (MAT1.LE.4) GO TO 2100
         WRITE(1,2030)
  2030   FORMAT(/' INVALID OPERAND'
     2            //' OPERATION NOT PERFORMED')
         GO TO 2999
  2100   WRITE(1,2110)
  2110   FORMAT(/' VALUE OF SCALAR ? ')
         READ(1,2120) SCALAR
  2120   FORMAT(F10.0)
         NROWS = NROWV(MAT1)
         NCOLS = NCOLV(MAT1)
         IF (MAT1.EQ.1) CALL SADD(SCALAR,NROWS,NCOLS,
        A,A)
         IF (MAT1.EQ.2) CALL SADD(SCALAR,NROWS,NCOLS,
        B,B)
         IF (MAT1.EQ.3) CALL SADD(SCALAR,NROWS,NCOLS,
        C,C)
         IF (MAT1.EQ.4) CALL SADD(SCALAR,NROWS,NCOLS,
        D,D)
```

```
   2999   CONTINUE
          GO TO 200
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *     MATRIX ADDITION                          *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   3000   WRITE(1,3010)
   3010   FORMAT(/' MATRIX ADDITION'
      2            /' A = A + B'
      3            //' NUMBER OF MATRIX A ? ')
          READ(1,3020) MAT2
   3020   FORMAT(I1)
          WRITE(1,3030)
   3030   FORMAT(' NUMBER OF MATRIX B ? ')
          READ(1,3020) MAT1
          IF ( (MAT1.LE.4).AND.(MAT2.LE.4)
      2       .AND.(NROWV(MAT1).EQ.NROWV(MAT2) )
      3       .AND.(NCOLV(MAT1).EQ.NCOLV(MAT2) ) )
      4       GO TO 3100
          WRITE(1,3040)
   3040   FORMAT(/' INVALID OPERAND'
      2            /' OPERATION NOT PERFORMED')
          GO TO 3999
   3100   NROWS = NROWV(MAT1)
          NCOLS = NCOLV(MAT1)
          IF (MAT1.EQ.1) CALL MOVE(NROWS,NCOLS,A,X)
          IF (MAT1.EQ.2) CALL MOVE(NROWS,NCOLS,B,X)
          IF (MAT1.EQ.3) CALL MOVE(NROWS,NCOLS,C,X)
          IF (MAT1.EQ.4) CALL MOVE(NROWS,NCOLS,D,X)
          IF (MAT2.EQ.1) CALL MADD(NROWS,NCOLS,X,A,A)
          IF (MAT2.EQ.2) CALL MADD(NROWS,NCOLS,X,B,B)
          IF (MAT2.EQ.3) CALL MADD(NROWS,NCOLS,X,C,C)
          IF (MAT2.EQ.4) CALL MADD(NROWS,NCOLS,X,D,D)
   3999   CONTINUE
          GO TO 200
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *     SCALAR MULTIPLICATION                    *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   4000   WRITE(1,4010)
   4010   FORMAT(/' SCALAR MULTIPLY'
      2            //' MATRIX NUMBER ? ')
          READ(1,4020) MAT1
   4020   FORMAT(I1)
```

```
          IF  (MAT1.LE.4)  GO TO 4100
          WRITE(1,4030)
4030      FORMAT(/' INVALID OPERAND'
     2              /' OPERATION NOT PERFORMED')
          GO TO 4999
4100      WRITE(1,4110)
4110      FORMAT(/' VALUE OF SCALAR  ? ')
          READ(1,4120) SCALAR
4120      FORMAT(F10.0)
          NROWS = NROWV(MAT1)
          NCOLS  = NCOLV(MAT1)
          IF  (MAT1.EQ.1)  CALL  SMULT(SCALAR,NROWS,NCOLS,
          A,A)
          IF  (MAT1.EQ.2)  CALL  SMULT(SCALAR,NROWS,NCOLS,
          B,B)
          IF  (MAT1.EQ.3)  CALL  SMULT(SCALAR,NROWS,NCOLS,
          C,C)
          IF  (MAT1.EQ.4)  CALL  SMULT(SCALAR,NROWS,NCOLS,
          D,D)
4999      CONTINUE
          GO TO 200
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     *     MATRIX MULTIPLICATION                      *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
5000      WRITE(1,5010)
5010      FORMAT(/' MATRIX MULTIPLICATION'
     2              /' A = B * C'
     3              //' NUMBER OF MATRIX  A ? ')
          READ(1,5020) MAT3
5020      FORMAT(I1)
          WRITE(1,5030)
5030      FORMAT(' NUMBER OF MATRIX B ? ')
          READ(1,5020) MAT1
          WRITE(1,5040)
5040      FORMAT(' NUMBER OF MATRIX C ? ')
          READ(1,5020) MAT2
          NROWS  = NROWV(MAT1)
          NCOLS   = NCOLV(MAT1)
          NROWS2 = NROWV(MAT2)
          NCOLS2 = NCOLV(MAT2)
          IF  ( (MAT1.LE.4).AND.(MAT2.LE.4).AND.(MAT3.LE.4)
```

```
    2        .AND.(NCOLS.EQ.NROWS2) ) GO TO 5100
         WRITE(1,5050)
5050     FORMAT(/' INVALID OPERAND'
    2            /' OPERATION NOT PERFORMED')
         GO TO 5999
5100     IF (MAT1.EQ.1) CALL MOVE(NROWS,NCOLS,A,X)
         IF (MAT1.EQ.2) CALL MOVE(NROWS,NCOLS,B,X)
         IF (MAT1.EQ.3) CALL MOVE(NROWS,NCOLS,C,X)
         IF (MAT1.EQ.4) CALL MOVE(NROWS,NCOLS,D,X)
         IF (MAT2.EQ.1)
    2       CALL MMULT(NROWS,NCOLS,NCOLS2,X,A,Y)
         IF (MAT2.EQ.2)
    2       CALL MMULT(NROWS,NCOLS,NCOLS2,X,B,Y)
         IF (MAT2.EQ.3)
    2       CALL MMULT(NROWS,NCOLS,NCOLS2,X,C,Y)
         IF (MAT2.EQ.4)
    2       CALL MMULT(NROWS,NCOLS,NCOLS2,X,D,Y)
         IF (MAT3.EQ.1) CALL MOVE(NROWS,NCOLS2,Y,A)
         IF (MAT3.EQ.2) CALL MOVE(NROWS,NCOLS2,Y,B)
         IF (MAT3.EQ.3) CALL MOVE(NROWS,NCOLS2,Y,C)
         IF (MAT3.EQ.4) CALL MOVE(NROWS,NCOLS2,Y,D)
         NROWV(MAT3) = NROWS
         NCOLV(MAT3) = NCOLS2
5999     CONTINUE
         GO TO 200
C    *************************************
C    *      TRANSPOSITION                *
C    *************************************
6000     WRITE(1,6010)
6010     FORMAT(/' TRANSPOSITION'
    2            /' A = TRN(B)'
    3           //' NUMBER OF MATRIX A ? ')
         READ(1,6020) MAT2
6020     FORMAT(I1)
         WRITE(1,6030)
6030     FORMAT(' NUMBER OF MATRIX B ? ')
         READ(1,6020) MAT1
         IF (MAT1.LE.4.AND.MAT2.LE.4) GO TO 6100
         WRITE(1,6040)
6040     FORMAT(/' INVALID RESPONSE'
    2            /' OPERATION NOT PERFORMED')
```

```
          GO TO 6999
   6100    NROWS = NROWV(MAT1)
           NCOLS  = NCOLV(MAT1)
           IF  (MAT1.EQ.1)  CALL  TRAN(NROWS,NCOLS,A,X)
           IF  (MAT1.EQ.2)  CALL  TRAN(NROWS,NCOLS,B,X)
           IF  (MAT1.EQ.3)  CALL  TRAN(NROWS,NCOLS,C,X)
           IF  (MAT1.EQ.4)  CALL  TRAN(NROWS,NCOLS,D,X)
           IF  (MAT2.EQ.1)  CALL  MOVE(NCOLS,NROWS,X,A)
           IF  (MAT2.EQ.2)  CALL  MOVE(NCOLS,NROWS,X,B)
           IF  (MAT2.EQ.3)  CALL  MOVE(NCOLS,NROWS,X,C)
           IF  (MAT2.EQ.4)  CALL  MOVE(NCOLS,NROWS,X,D)
           NROWV(MAT2) = NCOLS
           NCOLV(MAT2)  = NROWS
   6999    CONTINUE
           GO TO 200
C    ************************************
C    *      INVERSION                   *
C    ************************************
   7000    WRITE(1,7010)
   7010    FORMAT(/' MATRIX INVERSION'
      2               /'  A  =  INV(B)'
      3              //' NUMBER OF MATRIX A  ?  ')
           READ(1,7020)  MAT2
   7020    FORMAT(I1)
           WRITE(1,7030)
   7030    FORMAT(' NUMBER OF MATRIX B  ?  ')
           READ(1,7020)  MAT1
           NROWS = NROWV(MAT1)
           NCOLS  = NCOLV(MAT1)
           IF  ( (MAT1.LE.4).AND.(NROWS.EQ.NCOLS) )
      2        GO TO 7100
           WRITE(1,7040)
   7040    FORMAT(/' INVALID OPERAND'
      2               /' OPERATION NOT PERFORMED')
           GO TO 7999
   7100    IF  (MAT1.EQ.1)  CALL  MINV(NROWS,A,X,DET)
           IF  (MAT1.EQ.2)  CALL  MINV(NROWS,B,X,DET)
           IF  (MAT1.EQ.3)  CALL  MINV(NROWS,C,X,DET)
           IF  (MAT1.EQ.4)  CALL  MINV(NROWS,D,X,DET)
           WRITE(1,7120)  DET
   7120    FORMAT(/' DETERMINANT  ',F15.5)
           IF  (DET.EQ.0.0)  GO TO 7999
```

```
        IF  (MAT2.EQ.1)  CALL  MOVE(NROWS,NROWS,X,A)
        IF  (MAT2.EQ.2)  CALL  MOVE(NROWS,NROWS,X,B)
        IF  (MAT2.EQ.3)  CALL  MOVE(NROWS,NROWS,X,C)
        IF  (MAT2.EQ.4)  CALL  MOVE(NROWS,NROWS,X,D)
        NROWV(MAT2)  =  NROWS
        NCOLV(MAT2)  =  NROWS
 7999   CONTINUE
        GO TO 200
C    ***********************************
C    *    MOVE                         *
C    ***********************************
 8000   WRITE(1,8010)
 8010   FORMAT(/' MOVE MATRIX'
     2          /' A = B'
     3          //' NUMBER FOR MATRIX A ? ')
        READ(1,8020) MAT2
 8020   FORMAT(I1)
        WRITE(1,8030)
 8030   FORMAT(' NUMBER FOR MATRIX B ? ')
        READ(1,8020) MAT1
        IF  (MAT1.LE.4)  GO  TO 8100
        WRITE(1,8040)
 8040   FORMAT(/' INVALID OPERAND'
     2          /' OPERATION NOT PERFORMED')
        GO TO 8999
 8100   NROWS = NROWV(MAT1)
        NCOLS  =  NCOLV(MAT1)
        IF  (MAT1.EQ.1)  CALL  MOVE(NROWS,NCOLS,A,X)
        IF  (MAT1.EQ.2)  CALL  MOVE(NROWS,NCOLS,B,X)
        IF  (MAT1.EQ.3)  CALL  MOVE(NROWS,NCOLS,C,X)
        IF  (MAT1.EQ.4)  CALL  MOVE(NROWS,NCOLS,D,X)
        IF  (MAT2.EQ.1)  CALL  MOVE(NROWS,NCOLS,X,A)
        IF  (MAT2.EQ.2)  CALL  MOVE(NROWS,NCOLS,X,B)
        IF  (MAT2.EQ.3)  CALL  MOVE(NROWS,NCOLS,X,C)
        IF  (MAT2.EQ.4)  CALL  MOVE(NROWS,NCOLS,X,D)
        NROWV(MAT2)  =  NROWS
        NCOLV(MAT2)  =  NCOLS
 8999   CONTINUE
        GO TO 200
C    ***********************************
C    *    READ                         *
C    ***********************************
```

```
   9000   WRITE(1,9010)
   9010   FORMAT(/' READ MATRIX'
      2           //' NUMBER OF MATRIX ? ')
           READ(1,9020) MAT1
   9020   FORMAT(I1)
           IF (MAT1.LE.4) GO TO 9100
           WRITE(1,9060)
   9060   FORMAT(/' INVALID PARAMETERS'
      2            /' OPERATION NOT PERFORMED')
           GO TO 9999
   9100   IF (MAT1.EQ.1) CALL MREAD(NROWS,NCOLS,A)
           IF (MAT1.EQ.2) CALL MREAD(NROWS,NCOLS,B)
           IF (MAT1.EQ.3) CALL MREAD(NROWS,NCOLS,C)
           IF (MAT1.EQ.4) CALL MREAD(NROWS,NCOLS,D)
           NROWV(MAT1) = NROWS
           NCOLV(MAT1) = NCOLS
   9999   CONTINUE
           GO TO 200
C   ************************************
C   *    WRITE                         *
C   ************************************
  10000   WRITE(1,10010)
  10010   FORMAT(/' WRITE MATRIX'
      2            //' NUMBER OF MATRIX ? ')
           READ(1,10020) MAT1
  10020   FORMAT(I1)
           NROWS = NROWV(MAT1)
           NCOLS = NCOLV(MAT1)
           IF ( (MAT1.LE.4).AND.(NROWS.LE.10)
      2       .AND.(NCOLS.LE.10) ) GO TO 10100
           WRITE(1,10030)
  10030   FORMAT(/' INVALID OPERAND'
      2            /' OPERATION NOT PERFORMED')
           GO TO 10999
  10100   WRITE(2,10110) MAT1
  10110   FORMAT(///' MATRIX ',I2)
           IF (MAT1.EQ.1) CALL MPRINT(NROWS,NCOLS,A)
           IF (MAT1.EQ.2) CALL MPRINT(NROWS,NCOLS,B)
           IF (MAT1.EQ.3) CALL MPRINT(NROWS,NCOLS,C)
           IF (MAT1.EQ.4) CALL MPRINT(NROWS,NCOLS,D)
  10999   CONTINUE
           GO TO 200
```

```
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     *     FINAL MESSAGE                             *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
11000    WRITE(1,11010)
11010    FORMAT(/' END OF PROGRAM' /)
         STOP
         END
```

TEST RUN I

The following test run illustrates the matrix transposition, and matrix multiplication operations:

PROGRAM P0803

GENERAL-PURPOSE
MATRIX MANIPULATIONS.

GIVE COMMAND HELP
TO DISPLAY INSTRUCTIONS.

COMMAND ? HELP

```
COMMAND        EXPLANATION
HELP      DISPLAY INSTRUCTIONS
SADD      SCALAR ADDITION
MADD      MATRIX ADDITION
SMLT      SCALAR MULTIPLICATION
MMLT      MATRIX MULTIPLICATION
TRAN      TRANSPOSITION
INVT      INVERSION
MOVE      MOVE MATRIX
READ      READ MATRIX
WRIT      WRITE MATRIX
STOP      TERMINATE PROGRAM
          TYPE THE VALUE 1 TO CONTINUE   1
```

THE FOUR AVAILABLE MATRICES ARE
NUMBERED 1, 2, 3, AND 4. THE
COMMAND NAME CONTAINS FOUR LETTERS.

COMMAND ? READ

READ MATRIX

NUMBER OF MATRIX ? 1

ENTER VALUES FOR MATRIX

NUMBER OF ROWS   ? 2

NUMBER OF COLUMNS ? 3

VALUE FOR

ROW 1

    COL 1 ? 1

    COL 2 ? 2

    COL 3 ? 3

ROW 2

    COL 1 ? 4

    COL 2 ? 5

    COL 3 ? 6

COMMAND ? TRAN

TRANSPOSITION

A = TRN(B)

NUMBER OF MATRIX A ? 2

NUMBER OF MATRIX B ? 1

COMMAND ? WRIT

WRITE MATRIX

NUMBER OF MATRIX ? 2

COMMAND ? MMLT

MATRIX MULTIPLICATION

A = B * C

NUMBER OF MATRIX A ? 3

NUMBER OF MATRIX B ? 1

NUMBER OF MATRIX C ? 2

COMMAND ? WRIT

WRIT MATRIX

NUMBER OF MATRIX ? 3

COMMAND ? STOP

END OF PROGRAM

PRINTED OUTPUT

The following is the printed output from the test run:

MATRIX 2

CONTENTS OF MATRIX
ROW 1
    1.000000   4.000000

ROW 2
    2.000000   5.000000

ROW 3
    3.000000   6.000000

MATRIX 3

CONTENTS OF MATRIX
ROW 1
    14.000000   32.000000

ROW 2
    32.000000   77.000000

TEST RUN II

The following test run illustrates the move command, the scalar add and the scalar multiply:

PROGRAM P0803

GENERAL-PURPOSE MATRIX MANIPULATIONS.

GIVE COMMAND HELP TO DISPLAY INSTRUCTIONS.

COMMAND ? READ

READ MATRIX

NUMBER OF MATRIX ? 1

ENTER VALUES FOR MATRIX

```
NUMBER OF ROWS        ? 2
NUMBER OF COLUMNS ? 3
VALUE FOR
ROW 1
     COL  1 ? 1
     COL  2 ? 2
     COL  3 ? 3
ROW 2
     COL  1 ? 4
     COL  2 ? 5
     COL  3 ? 6
COMMAND ? WRIT
WRITE MATRIX
NUMBER OF MATRIX ? 1
COMMAND ? MOVE
MOVE MATRIX
A = B
NUMBER FOR MATRIX A ? 2
NUMBER FOR MATRIX B ? 1
COMMAND ? SADD
SCALAR ADDITION
MATRIX NUMBER ? 2
VALUE OF SCALAR ? 5
COMMAND ? WRIT
WRITE MATRIX
NUMBER OF MATRIX ? 2
COMMAND ? SMLT
SCALAR MULTIPLICATION
NUMBER OF MATRIX ? 2
VALUE OF SCALAR ? 2
COMMAND ? WRIT
WRITE MATRIX
```

NUMBER OF MATRIX ? 2

COMMAND ? STOP

END OF PROGRAM

PRINTED OUTPUT

The following printed output resulted from the test run:

MATRIX 1

CONTENTS OF MATRIX

ROW 1

1.000000	2.000000	3.000000

ROW 2

4.000000	5.000000	6.000000

MATRIX 2

CONTENTS OF MATRIX

ROW 1

6.000000	7.000000	8.000000

ROW 2

9.000000	10.000000	11.000000

MATRIX 3

CONTENTS OF MATRIX

ROW 1

12.000000	14.000000	16.000000

ROW 2

18.000000	20.000000	22.000000

TEST RUN III

This test run inverts a square matrix and tests the inverse by taking the product of the original matrix and its inverse:

PROGRAM P0803

GENERAL-PURPOSE MATRIX MANIPULATIONS.

```
GIVE COMMAND HELP
TO DISPLAY INSTRUCTIONS.
COMMAND ? READ
READ MATRIX
NUMBER OF MATRIX ? 1
ENTER VALUES FOR MATRIX
NUMBER OF ROWS      ? 2
NUMBER OF COLUMNS ? 2
VALUE FOR
ROW 1
    COL 1 ? 4
    COL 2 ? 2
ROW 2
    COL 1 ? 2
    COL 2 ? 4
COMMAND ? INVT
MATRIX INVERSION
A = INV(B)
NUMBER OF MATRIX A ? 2
NUMBER OF MATRIX B ? 1
DETERMINANT        12.00000
COMMAND ? MMLT
MATRIX MULTIPLICATION
A = B * C
NUMBER OF MATRIX A ? 3
NUMBER OF MATRIX B ? 1
NUMBER OF MATRIX C ? 2
COMMAND ? WRIT
WRITE MATRIX
NUMBER OF MATRIX ? 1
COMMAND ? WRIT
```

WRITE MATRIX

NUMBER OF MATRIX ? 2

COMMAND ? WRIT

WRITE MATRIX

NUMBER OF MATRIX ? 3

COMMAND ? STOP

END OF PROGRAM

PRINTED OUTPUT

The following printed output results from the test run:

MATRIX 1

CONTENTS OF MATRIX
ROW 1
      4.000000   2.000000
ROW 2
      2.000000   4.000000

MATRIX 2

CONTENTS OF MATRIX
ROW 1
      .333333   -.166667
ROW 2
      -.166667   .333333

MATRIX 3

CONTENTS OF MATRIX
ROW 1
      1.000000   0.000000
ROW 2
      0.000000   1.000000

8.4   Simultaneous Equations

SOLVING EQUATIONS

One of the tasks of elementary algebra consists of solving algebraic equations. The equation

$$a*x = b$$

has the solution

$$x = (1/a)*b.$$

The expression $1/a$ is the reciprocal of a and is called the multiplicative inverse of a.

SIMULTANEOUS EQUATIONS

Solving equations extends to systems of simultaneous equations. The following equations represent a system of three simultaneous linear equations in three unknowns:

$$3X1 + 5X2 - 7X3 = 20$$
$$4X1 - 2X2 + 4X3 = 16$$
$$X1 + 2X1 + 3X2 = 24$$

A unique solution may exist for sets of simultaneous linear equations if the number of equations exactly equals the number of unknowns. The solution does not always exist. The equation

$$a*x = b$$

does not have a solution if $a=0$. A system of simultaneous equations may not have a solution for similar reasons.

MATRIX FORMULATION.

Matrix algebra provides a simple representation for the set of equations. Let the matrix A contain the coefficients on the left.

$$A = \begin{matrix} 3 & 5 & -7 \\ 4 & -2 & 4 \\ 1 & 5 & 3 \end{matrix}$$

The matrix B is a column vector containing the constants from the right as its only column.

$$B = \begin{matrix} 20 \\ 16 \\ 24 \end{matrix}$$

The matrix X is a column vector containing the values for the unknowns as its only column:

$$X = \begin{matrix} X1 \\ X2 \\ X3 \end{matrix}$$

The matrix equation

$$A * X = B$$

represents the simultaneous linear equations.

MATRIX SOLUTION

Pre-multiplying both sides of the matrix equation by the inverse of A gives

$$X = \mathrm{Inv}(A) * B$$

which solves for the values of the unknowns. If the matrix A is singular (determinant zero) there is no unique solution. This corresponds to $a=0$ in the equation

$$a*x = b.$$

PROGRAM

The following program uses the subroutines of the previous sections to perform the processing steps for solving a set of simultaneous linear equations:

```
      PROGRAM P0804
C   ***********************************
C   *    P0804                        *
C   ***********************************
```

```
C     AUTHOR
C          COPYRIGHT 1982
C          BY LAWRENCE MCNITT.
C     PURPOSE
C          SOLVE SIMULTANEOUS
C          LINEAR EQUATIONS.
C     SYSTEM
C          MICROSOFT FORTRAN
C          RADIO SHACK TRS-80.
C     ****************************************
C     *     ORGANIZATION                     *
C     ****************************************
C     INITIAL MESSAGE
C     INPUT
C     PROCESS
C          S08F: MATRIX INVERSION SUBROUTINE
C     OUTPUT
C     FINAL MESSAGE
C     ****************************************
C     *     VARIABLES                        *
C     ****************************************
C     A(10,10)     MATRIX OF COEFFICIENTS
C     V(10,10)     INVERSE OF A
C     B(10,10)       CONSTANTS FOR EQUATION AX=B
C     X(10,10)       SOLUTION VALUES
C     NSIZE        NUMBER OF EQUATIONS AND UNKNOWNS
C     DET          DETERMINANT
C     I            ROW SUBSCRIPT
C     J            COLUMN SUBSCRIPT
C     IRESP        USER RESPONSE
C     ****************************************
C     *     INITIAL MESSAGE                  *
C     ****************************************
          DIMENSION A(10,10),B(10,10),V(10,10),X(10,10)
          WRITE(1,110)
    110   FORMAT(/' PROGRAM P0804'
       2            //' SOLVE A SET OF N SIMULTANEOUS'
       3             /' LINEAR EQUATIONS IN N UNKNOWNS.')
C     ****************************************
C     *     INPUT                            *
C     ****************************************
    200   WRITE(1,210)
```

```
210    FORMAT(/' NUMBER OF EQUATIONS ? ')
       READ(1,220) NSIZE
220    FORMAT(I2)
       WRITE(1,230)
230    FORMAT(' VALUES FOR' /)
       DO 290 I = 1, NSIZE
           WRITE(1,240) I
240        FORMAT(' EQUATION ',I2/)
           DO 270 J = 1, NSIZE
               WRITE(1,250) J
250            FORMAT('   VAR ',I2,' ? ')
               READ(1,260) A(I,J)
260            FORMAT(F10.0)
270        CONTINUE
           WRITE(1,280)
280        FORMAT('   CONSTANT ? ')
           READ(1,285) B(I,1)
285        FORMAT(F10.0)
290    CONTINUE
C  *************************************
C  *    PROCESS                        *
C  *************************************
       CALL  MINV(NSIZE,A,V,DET)
       WRITE(1,310) DET
310    FORMAT(/' DETERMINANT = ',F15.5)
       IF (DET.EQ.0.0) GO TO 500
       CALL  MMULT(NSIZE,NSIZE,1,V,B,X)
C  *************************************
C  *    OUTPUT                         *
C  *************************************
       WRITE(1,410)
410    FORMAT(/' VAR        VALUE')
       DO 430 I = 1, NSIZE
           WRITE(1,420) I,X(I,1)
420        FORMAT(2X,I2,F15.5)
430    CONTINUE
C  *************************************
C  *    FINAL MESSAGE                  *
C  *************************************
500    WRITE(1,510)
510    FORMAT(/' TRY ANOTHER PROBLEM (Y/N) ? ')
       READ(1,520) IRESP
```

```
520    FORMAT(A1)
       IF (IRESP.EQ.'Y') GO TO 200
       WRITE(1,530)
530    FORMAT(/' END OF PROGRAM/')
       STOP
       END
```

TEST RUN

The following test run resulted from running the program:

PROGRAM P0804

SOLVE A SET OF N SIMULTANEOUS
LINEAR EQUATIONS IN N UNKNOWNS.

NUMBER OF EQUATIONS ? 2

VALUES FOR

EQUATION 1

    VAR 1 ? 4

    VAR 2 ? 2

    CONSTANT ? 20

EQUATION 2

    VAR 1 ? 2

    VAR 2 ? 4

    CONSTANT ? 30

DETERMINANT   12.00000

VAR	VALUE
1	1.66667
2	6.66667

TRY ANOTHER PROBLEM (Y/N) ? N

END OF PROGRAM

8.5   Exercises

1. Implement the general-purpose matrix manipulation program together with its subroutines.

2. Implement the program and its subroutines for solving simul-
   taneous linear equations.

3. Use the random number function to generate a set of values
   for a 10-row by 10-column matrix. Invert that matrix using
   the matrix inversion subroutine.

4. Use matrix multiplication methods to determine the total
   labor cost per job given the following matrix of times by each
   worker on each job and the column vector of hourly payrates
   for the workers:

TIME IN HOURS				PAY RATE	
Job	Employee			Employee	Rate
	1	2	3	1	8.75
1	12	0	15	2	6.25
2	8	20	6	3	4.80
3	10	8	12		
4	0	15	18		

5. For an inverse to exist, a matrix must be square and nonsin-
   gular. A singular matrix will result if two rows are equal or if
   one is a multiple of another. The following matrix is singular:

$$
\begin{matrix}
3 & 4 & -5 \\
3 & 4 & -5 \\
2 & 1 & 3
\end{matrix}
$$

Test the matrix inversion routine with this matrix.

9 Random files

OVERVIEW Random files overcome some of the deficiencies of sequential files. Records can be assessed in any order, and can be updated in place. This eliminates the need of processing the entire file in order to change a few records. Random files are not perfect. Processing the entire random file takes longer than processing the entire corresponding sequential file.

☐

9.1   Relative Access

RELATIVE FILE ORGANIZATION

Microsoft FORTRAN for the Radio Shack TRS-80 provides random access methods for disk files. Sequential disk files include an end-of-record code for each record. This adds one byte to the record size. Random access files do not include this end-of-record code. Random access files append a special end-of-file record at the end of the file.

STORAGE ALLOCATION

Relative files should consist of one extent on the disk drive. An extent is a contiguous region of the disk. Sequential files may consist of several extents scattered over the disk. An extent contains one or more granules. A granule for the Radio Shack Model III contains three sectors of 256 bytes each. File access methods for sequential files will automatically link from extent to extent. This linking of noncontiguous extents is not practical for relative files. The CREATE utility creates a pre-allocated region for the file specifying the record size and the number of records.

205

FILE OPERATIONS

Necessary file operations include initializing the file, inserting and deleting records, changing values for existing records, and inspecting the contents of records. This chapter illustrates these operations using several example programs.

STATISTICAL DATA BASE

The programs of this chapter illustrate the development and use of a general-purpose statistical data base system for creating and maintaining a file of numerical data. A previous chapter used sequential files for this purpose. Those programs loaded the entire file into memory for modifications.

Using random files for the statistical data base eliminates the need of loading the entire file. Updating individual records from the terminal is fast and simple. There is a penalty in the processing speed for those programs that scan the entire random file. Reading an entire sequential file is faster than reading an entire random file.

SPECIFICATION FILE

The system includes a specification file giving the maximum number of records for the file and the labels identifying the variables. These labels can be added to printed output to enhance readability.

The specification file is sequential. Each record contains 13 bytes including the end-of-record code. The first record contains the maximum size of the file. The first five positions give the maximum number of records. The next five positions are unused. The next two bytes give the number of variables for the file.

The following records give the labels in the 3A4 format. There is a separate record for each label. The 16th byte is the end-of-record marker.

DATA FILE

The data file contains one record for each data record. There may be up to 64 values per record. Each record contains a two-

byte code giving the record number (relative location) for each valid record in the file. A negative record number signifies an empty record. If NVARS gives the number of variables, the expression

$$LENGTH = 2 + 4 * NVARS$$

gives the number of bytes per record.

All quantities are unformatted. This eliminates the need for number conversion for input/output with the data file. The record number is a two-byte integer. Each data value is a four-byte real value.

INITIALIZING THE FILES

The following program initializes the specification file and the data file for the statistical data base:

```
            PROGRAM P0901
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *    P0901                                           *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         INITIALIZE SPECIFICATION FILE
C         AND DISK DATA FILE REGION.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80.
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    *    ORGANIZATION                                    *
C    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C    INITIAL MESSAGE
C    GET FILE NAMES
C    GET FILE SIZE
C    GET VARIABLE NAMES
C    SPECIFICATION FILE
C    DATA FILE
C    END OF PROGRAM
```

```
C     ****************************************
C     *     VARIABLES                        *
C     ::::::::::::::::::::::::::::::::::::::::::::
C     VNAMES(64,3)  VARIABLE NAMES
C     VALUES(64)    INITIAL DATA VALUES
C     FNAME1(4)     SPECIFICATION FILE NAME
C     FNAMES2(4)    DATA FILE NAME
C     MAXOBS    MAXIMUM NUMBER OF OBSERVATIONS
C     IREC      CURRENT RECORD NUMBER
C     NIREC     NEGATIVE OF CURRENT RECORD NUMBER
C     NVARS     NUMBER OF VARIABLES
C     IVAR      CURRENT VARIABLE
C     LENGTH    NUMBER OF BYTES PER RECORD
C     J         INDEX
C     ****************************************
C     *     INITIAL MESSAGE                  *
C     ****************************************
          DIMENSION VNAMES(64,3),VALUES(64),
     2          FNAME1(4),FNAME2(4)
          WRITE(1,110)
  110     FORMAT(/' INITIALIZE SPECIFICATION FILE'
     2              /' AND DISK DATA FILE REGION.')
C     ****************************************
C     *     GET FILE NAMES                   *
C     ****************************************
          WRITE(1,1010)
 1010     FORMAT(/' NAME OF DISK DATA FILE ? ')
          READ(1,1020) FNAME2
 1020     FORMAT(4A4)
          WRITE(1,1030)
 1030     FORMAT(' NAME OF SPECIFICATION FILE ? ')
          READ(1,1020) FNAME1
C     ****************************************
C     *     GET FILE SIZE                    *
C     ****************************************
          WRITE(1,2010)
 2010     FORMAT(' MAXIMUM NUMBER OF OBSERVATIONS ? ')
          READ(1,2020) MAXOBS
 2020     FORMAT(I5)
          WRITE(1,2030)
 2030     FORMAT(' NUMBER OF VARIABLES ? ')
          READ(1,2040) NVARS
 2040     FORMAT(I2)
```

```
C     ************************************
C     *       GET VARIABLE NAMES                *
C     ************************************
          WRITE(1,3010)
   3010    FORMAT(' VARIABLE NAMES MAY HAVE'
       2            /' UP TO 12 CHARACTERS EACH'
       3            //' NAME FOR' /)
          DO 3040 IVAR = 1, NVARS
             WRITE(1,3020) IVAR
   3020       FORMAT(' VAR ',I2,' ? ')
             READ(1,3030) (VNAMES(IVAR,J),J=1,3)
   3030       FORMAT(3A4)
   3040    CONTINUE
C     ************************************
C     *       SPECIFICATION FILE                *
C     ************************************
          LENGTH = 13
          CALL OPEN(6,FNAME1,LENGTH)
          WRITE(6,4010) MAXOBS,NVARS
   4010    FORMAT(I5,5X,I2)
          DO 4030 IVAR = 1, NVARS
             WRITE(6,4020) (VNAMES(IVAR,J),J=1,3)
   4020       FORMAT(3A4)
   4030    CONTINUE
          ENDFILE 6
C     ************************************
C     *       DATA FILE                         *
C     ************************************
          LENGTH = 4 * NVARS + 2
          CALL OPEN(7,FNAME2,LENGTH)
          DO 5010 IVAR = 1, NVARS
             VALUES(IVAR) = 0.0
   5010    CONTINUE
          DO 5020 IREC = 1, MAXOBS
             NIREC = -IREC
             WRITE(7,REC=IREC) NIREC,
       2            (VALUES(IVAR),IVAR=1,NVARS)
   5020    CONTINUE
          ENDFILE 7
C     ************************************
C     *       END OF PROGRAM                    *
C     ************************************
          WRITE(1,9010)
```

```
   9010   FORMAT(/' END OF PROGRAM' /)
          STOP
          END
```

TEST RUN

The following test run initializes the specification file SPEC/DAT and the data file FILE/DAT:

INITIALIZE SPECIALIZATION FILE
AND DISK DATA FILE REGION.

NAME OF DISK DATA FILE ? FILE/DAT

NAME OF SPECIFICATION FILE ? SPEC/DAT

MAXIMUM NUMBER OF OBSERVATIONS ? 10

NUMBER OF VARIABLES ? 3

VARIABLE NAMES MAY HAVE
UP TO 12 CHARACTERS EACH

NAME FOR

VAR 1 ? ERRORS

VAR 2 ? AGE

VAR 3 ? TIME

END OF PROGRAM

9.2   Update in Place

RANDOM ACCESS

Random access files allow updating in place. The record is loaded from the disk, changed, and written back to its prior location. This feature eliminates the need for processing the entire file when only a small fraction of the records need attention.

FILE MAINTENANCE

File maintenance operations include inserting new records, deleting old records, and changing values associated with existing records. This requires that the system recognize when any record location contains data and when it is empty of data.

The method chosen is to include a record number with each record location in the file. A positive value signifies valid data. A negative value signifies an empty record location.

PROGRAM

The following program performs the file maintenance operations for the statistical data base system:

```
            PROGRAM P0902
C     *************************************
C     *    P0902                          *
C     *************************************
C     AUTHOR
C          COPYRIGHT 1982
C          BY LAWRENCE MCNITT.
C     PURPOSE
C          UPDATE CONTENTS OF
C          DISK DATA FILE.
C     SYSTEM
C          MICROSOFT FORTRAN
C          RADIO SHACK TRS-80.
C     *************************************
C     *    ORGANIZATION                   *
C     *************************************
C     INITIAL MESSAGE
C     SPECIFICATION FILE
C     MENU
C     ADD NEW OBSERVATIONS
C     CHANGE VALUES
C     DELETE OBSERVATIONS
C     LIST VALUES
C     END OF PROGRAM
C     *************************************
C     *    VARIABLES                      *
C     *************************************
C     VNAMES(64,3)  VARIABLE NAMES
C     VALUES(64)    VALUES FOR CURRENT RECORD
C     FNAME1(4)     NAME OF SPECIFICATION FILE
C     FNAME2(4)     NAME OF DATA FILE
C     MAXOBS     MAXIMUM NUMBER OF OBSERVATIONS
```

```
C    NVARS     NUMBER OF VARIABLES
C    IVAR      CURRENT VARIABLE
C    IREC      CURRENT RECORD NUMBER
C    NIREC     NEGATIVE OF RECORD NUMBER IF EMPTY
C    J         INDEX
C    LENGTH    NUMBER OF BYTES IN RECORD
C    IRESP     USER RESPONSE
C    NADD      NUMBER OF OBSERVATIONS TO ADD
C    ISTART    STARTING RECORD NUMBER
C    ISTOP     LAST RECORD NUMBER
C    ICODE     CHANGE CODE
C    ***********************************
C    *    INITIAL MESSAGE              *
C    ***********************************
         DIMENSION VNAMES(64,3),VALUES(64),
     2             FNAME1(4),FNAME2(4)
         WRITE(1,110)
110      FORMAT(/' PROGRAM P0902'
     2            //' UPDATE DISK'
     3             /' DATA FILE.')
C    ***********************************
C    *    SPECIFICATION FILE           *
C    ***********************************
         WRITE(1,210)
210      FORMAT(/' SPECIFICATION FILE NAME ? ')
         READ(1,220) FNAME1
220      FORMAT(4A4)
         WRITE(1,230)
230      FORMAT(' DATA FILE NAME ? ')
         READ(1,220) FNAME2
         LENGTH = 13
         CALL OPEN(6,FNAME1,LENGTH)
         READ(6,240) MAXOBS, NVARS
240      FORMAT(I5,5X,I2)
         DO 260 IVAR = 1, NVARS
            READ(6,250) (VNAMES(IVAR,J),J=1,3)
250         FORMAT(3A4)
260      CONTINUE
         ENDFILE 6
         LENGTH = 2 + 4 * NVARS
         CALL OPEN(7,FNAME2,LENGTH)
```

```
C    **************************************
C    *    MENU                            *
C    **************************************
     300    WRITE(1,310)
     310    FORMAT(/' OPTIONS'
        1           /'    1    ADD NEW OBSERVATIONS'
        2           /'    2    CHANGE EXISTING VALUES'
        3           /'    3    DELETE OBSERVATIONS'
        4           /'    4    LIST VALUES FOR OBSERVATION'
        5           /'    5    TERMINATE PROCESSING'
        6          //' OPTION NUMBER ? ')
            READ(1,320) IRESP
     320    FORMAT(I1)
            GO TO (1000,2000,3000,4000,5000), IRESP
            WRITE(1,330)
     330    FORMAT(/' INVALID OPTION NUMBER')
            GO TO 300
C    **************************************
C    *    ADD NEW OBSERVATIONS            *
C    **************************************
     1000   WRITE(1,1010)
     1010   FORMAT(/' ADD NEW OBSERVATIONS'
        2          //' NUMBER OF OBSERVATIONS TO ADD ? ')
            READ(1,1020) NADD
     1020   FORMAT(I5)
            WRITE(1,1030)
     1030   FORMAT(' STARTING OBSERVATION ? ')
            READ(1,1040) ISTART
     1040   FORMAT(I5)
            ISTOP = ISTART + NADD - 1
            DO 1150 IREC = ISTART,ISTOP
               WRITE(1,1050) IREC
     1050      FORMAT(/' OBSERVATION ',I5)
               READ(7,REC=IREC) NIREC
               IF (NIREC.LT.0) GO TO 1100
     1060      WRITE(1,1070)
     1070      FORMAT(/' RECORD ALREADY CONTAINS VALUES.'
        2              /' DO YOU WANT TO OVERRIDE THEM (Y/N)
                          ? ')
               READ(1,1080) IRESP
     1080      FORMAT(A1)
```

```
               IF  (IRESP.EQ.'Y')  GO TO 1100
               IF  (IRESP.EQ.'N')  GO TO 1150
               WRITE(1,1090)
     1090      FORMAT(/' INVALID RESPONSE')
               GO TO 1060
     1100      WRITE(1,1110)
     1110      FORMAT(/' VALUE FOR' /)
               DO 1140 IVAR = 1, NVARS
                   WRITE(1,1120)  (VNAMES(IVAR,J),J=1,3)
     1120          FORMAT(1X,3A4,' ? ')
                   READ(1,1130) VALUES(IVAR)
     1130          FORMAT(F10.0)
     1140      CONTINUE
               WRITE(7,REC=IREC) IREC,
         2          (VALUES(IVAR),IVAR=1,NVARS)
     1150   CONTINUE
               GO TO 300
C   ***********************************
C   *   CHANGE VALUES                 *
C   ***********************************
     2000   WRITE(1,2010)
     2010   FORMAT(/' CHANGE EXISTING VALUES'
         2          /' OBSERVATION NUMBER ? ')
             READ(1,2020) IREC
     2020   FORMAT(I5)
             READ(7,REC=IREC) NIREC,
         2          (VALUES(IVAR),IVAR=1,NVARS)
             IF  (NIREC.GT.0) GO TO 2040
             WRITE(1,2030)
     2030   FORMAT(/' OBSERVATION NOT INSERTED YET'
         2          /' USE THE ROUTINE TO ADD'
         3          /' OBSERVATIONS TO THE FILE.')
             GO TO 2110
     2040   DO 2100 IVAR = 1, NVARS
               WRITE(1,2050)  (VNAMES(IVAR,J),J=1,3),
         2             VALUES(IVAR)
     2050      FORMAT(1X,3A4,F15.5)
             WRITE(1,2060)
     2060      FORMAT(' CHANGE VALUE  (Y/N) ? ')
             READ(1,2070) ICODE
     2070      FORMAT(A1)
             IF  (ICODE.EQ.'N') GO TO 2100
```

```
            WRITE(1,2080)
 2080       FORMAT(' NEW VALUE ? ')
            READ(1,2090) VALUES(IVAR)
 2090       FORMAT(F10.0)
 2100    CONTINUE
            WRITE(7,REC=IREC) IREC,
       2        (VALUES(IVAR),IVAR=1,NVARS)
 2110    WRITE(1,2120)
 2120    FORMAT(/' CHANGE ANOTHER OBSERVATION  (Y/N)
            ?')
            READ(1,2130) IRESP
 2130    FORMAT(A1)
            IF (IRESP.EQ.'Y') GO TO 2000
            GO TO 300
C   ***********************************
C   *    DELETE OBSERVATIONS               *
C   ***********************************
 3000    WRITE(1,3010)
 3010    FORMAT(/' DELETE OBSERVATION ? ')
            READ(1,3020) IREC
 3020    FORMAT(I5)
            READ(7,REC=IREC) NIREC,
       2        (VALUES(IVAR),IVAR=1,NVARS)
            IF (NIREC.GT.0) GO TO 3040
            WRITE(1,3030)
 3030    FORMAT(' RECORD DOES NOT CONTAIN DATA'
       2        /' NO NEED TO DELETE')
            GO TO 3100
 3040    WRITE(1,3050)
 3050    FORMAT(/'    VARIABLE        VALUE')
            DO 3070 IVAR = 1, NVARS
                WRITE(1,3060) (VNAMES(IVAR,J),J=1,3),
       2            VALUES(IVAR)
 3060       FORMAT(1X,3A4,F15.5)
 3070    CONTINUE
            WRITE(1,3080)
 3080    FORMAT(/' STILL WANT TO DELETE  (Y/N) ? ')
            READ(1,3090) ICODE
 3090    FORMAT(A1)
            IF (ICODE.EQ.'N') GO TO 3100
            NIREC = -IREC
            WRITE(7,REC=IREC) NIREC,
```

```
      2          (VALUES(IVAR),IVAR=1,NVARS)
 3100     WRITE(1,3110)
 3110     FORMAT(/' DELETE ANOTHER RECORD (Y/N ? ')
          READ(1,3120) IRESP
 3120     FORMAT(A1)
          IF (IRESP.EQ.'Y') GO TO 3000
          GO TO 300
C    **********************************
C    *    LIST VALUES                 *
C    **********************************
 4000     WRITE(1,4010)
 4010     FORMAT(/' LIST VALUES OF VARIABLES'
      2         /' FOR OBSERVATION ? ')
          READ(1,4020) IREC
 4020     FORMAT(I5)
          READ(7,REC=IREC) NIREC,
      2          (VALUES(IVAR),IVAR=1,NVARS)
          IF (NIREC.GT.0) GO TO 4040
          WRITE(1,4030)
 4030     FORMAT(' OBSERVATION IS EMPTY')
          GO TO 4070
 4040     DO 4060 IVAR = 1, NVARS
             WRITE(1,4050) (VNAMES(IVAR,J),J=1,3),
      2          VALUES(IVAR)
 4050        FORMAT(1X,3A4,F15.5)
 4060     CONTINUE
 4070     WRITE(1,4080)
 4080     FORMAT(/' LIST CONTENTS OF ANOTHER'
      2          /' OBSERVATION (Y/N) ? ')
          READ(1,4090) IRESP
 4090     FORMAT(A1)
          IF (IRESP.EQ.'Y') GO TO 4000
          GO TO 300
C    **********************************
C    *    END OF PROGRAM              *
C    **********************************
 5000     ENDFILE 7
          WRITE(1,5010)
 5010     FORMAT(/' END OF PROGRAM')
          STOP
          END
```

TEST RUN

The following set of test data is used for the test run:

Errors	Age	Time
12	2	6
15	4	6
18	5	4
19	3	2
22	6	3

The following test run illustrates the initial data entry and steps used in modifying the data file:

PROGRAM P0902

UPDATE DISK
DATA FILE.

SPECIFICATION FILE NAME ? SPEC/DAT

DATA FILE NAME ? FILE/DAT

OPTIONS
    1   ADD NEW OBSERVATIONS
    2   CHANGE EXISTING VALUES
    3   DELETE OBSERVATIONS
    4   LIST VALUES OBSERVATION
    5   TERMINATE PROCESSING

OPTION NUMBER ? 1

ADD NEW OBSERVATIONS

NUMBER OF OBSERVATIONS TO ADD ? 5

STARTING OBSERVATION ? 1

OBSERVATION 1

VALUE FOR

ERRORS   ? 12

AGE   ? 2

TIME   ? 6

OBSERVATION 2

VALUE FOR

ERRORS   ? 15

AGE   ? 4

TIME   ? 6

OBSERVATION 3

VALUE FOR

ERRORS   ? 18

AGE   ? 5

TIME   ? 4

OBSERVATION 4

VALUE FOR

ERROR   ? 19

AGE   ? 3

TIME   ? 2

OBSERVATION 5

VALUE FOR

ERRORS   ? 22

AGE   ? 66

TIME   ? 3

OPTION
    1   ADD NEW OBSERVATIONS
    2   CHANGE EXISTING VALUES
    3   DELETE OBSERVATIONS
    4   LIST VALUES FOR OBSERVATION
    5   TERMINATE PROCESSING

OPTION NUMBER ? 4

LIST VALUES OF VARIABLES
FOR OBSERVATION ? 5

ERRORS   22.00000
AGE   66.00000
TIME   3.00000

LIST CONTENTS OF ANOTHER
OBSERVATION (Y/N)? N

OPTIONS
    1   ADD NEW OBSERVATIONS

```
         2    CHANGE EXISTING VALUES
         3    DELETE OBSERVATIONS
         4    LIST VALUES FOR OBSERVATION
         5    TERMINATE PROCESSING
    OPTION NUMBER ? 2

    CHANGE EXISTING VALUES

    OBSERVATION NUMBER ? 3

    ERRORS              22.00000
    CHANGE VALUE (Y/N) ? N

    AGE                 66.00000
    CHANGE VALUE (Y/N) ? Y

    NEW VALUE ? 6.0

    TIME                3.00000
    CHANGE VALUE (Y/N) ? N

    CHANGE ANOTHER OBSERVATION (Y/N) ? N

    OPTION
         1    ADD NEW OBSERVATIONS
         2    CHANGE EXISTING VALUES
         3    DELETE OBSERVATIONS
         4    LIST VALUES FOR OBSERVATION
         5    TERMINATE PROCESSING
    OPTION NUMBER ? 5

    END OF PROGRAM
```

9.3   Online Inquiry

ONLINE PROCESSING

Online processing involves interactive processing with control of
the steps from a computer terminal at the time of the processing.
This is one of the prime benefits of microcomputers. Large time-
sharing systems provide this capability, but at high cost per work
station. Microcomputers provide this capability at an affordable
cost per station. The tasks may include insertions, deletions, and
changes.

BATCH REPORTS

Traditional batch processing systems perform weekly or monthly
updates of the computer files. During or after the update runs,

detailed reports give the new record values as well as summary reports measuring trends and totals.

The detailed reports for large files can be voluminous. These serve as archival records for legal and reference purposes. The reports give the record values as of the last update.

FORTRAN and COBOL have been the primary languages used by large computers for batch processing. BASIC and other languages can also function in this mode. Although microcomputers typically run in an online mode, they also can operate in batch mode. Batch mode operation does not require operator intervention during the course of the run.

INTERACTIVE INQUIRY

Random access files and computer terminals offer an alternative to batch processing. Users can access the file with an inquiry program which displays the current record contents. The user need not wait for some distant computer to print the report at its convenience. The user sees the information displayed immediately. This reduces the need for detailed listings. It also makes much better use of the user's time.

ONLINE UPDATE

Interactive processing from computer terminals also allows updates to be made to the file at any time. The organization does not need to accumulate transaction data for the monthly or weekly update run. The update may be made once a day or several times a day. With frequent updating the computer files better reflect the current status of the organization.

Along with improved flexibility and responsiveness comes the need for greater control over access and update capability. *Information that is more readily accessible is more readily mishandled.*

REAL TIME SYSTEMS

A real time system updates the computer files at the time of the actual transaction. This may be considered as a batch of size one. Online updating in real time requires random access files. Online updating does not necessarily have to be done in real time. Real time systems, however, require online updating.

ONLINE INQUIRY SYSTEMS

Online inquiry systems are useful regardless of the frequency of updating. Updating may be in batch mode. If the files allow random access, then online inquiry is possible. The purpose of online inquiry is to allow the user to inspect the contents of the current computer file.

RESTRICTIONS

Information in the computer files is usually restricted. Selected employees are permitted to access sensitive computer files. Even more stringent restrictions are made on those to have update capability. Few employees are authorized to make changes to those files. Online inquiry programs provide read-only access to files for those who need information from the files but who are not permitted to make changes to the files. Read/write access is needed by those who are allowed to make changes.

PROGRAM

The following program illustrates an online inquiry program for the statistical data base:

```
          PROGRAM P0903
C    ****************************************
C    *     P0903                            *
C    ****************************************
C    AUTHOR
C         COPYRIGHT 1982
C         BY LAWRENCE MCNITT.
C    PURPOSE
C         ONLINE INQUIRY FOR
C         DISK DATA FILE.
C    SYSTEM
C         MICROSOFT FORTRAN
C         RADIO SHACK TRS-80.
C    ****************************************
C    *     ORGANIZATION                     *
C    ****************************************
C    INITIAL MESSAGE
C    SPECIFICATION FILE
```

```
C    PROCESS
C    END OF PROGRAM
C    ***********************************
C    *    VARIABLES                    *
C    ***********************************
C    VNAMES(64,3)  VARIABLE NAMES
C    VALUES(64)    VALUES FOR RECORD
C    FNAME1(4)     NAME OF SPECIFICATION FILE
C    FNAME2(4)     NAME OF DATA FILE
C    MAXOBS    MAXIMUM NUMBER OF OBSERVATIONS
C    NVARS     NUMBER OF VARIABLES
C    IVAR      CURRENT VARIABLE
C    IREC      CURRENT RECORD
C    NIREC     NEGATIVE OF CURRENT RECORD IF EMPTY
C    J         INDEX
C    LENGTH    LENGTH OF RECORD IN BYTES
C    IRESP     USER RESPONSE
C    ***********************************
C    *    INITIAL MESSAGE              *
C    ***********************************
         DIMENSION VNAMES(64,3),VALUES(64),
      2            FNAME1(4),FNAME2(4)
         WRITE(1,110)
  110    FORMAT(/' PROGRAM P0903'
      2           //' ONLINE INQUIRY SYSTEM'
      3           /' FOR DISK DATA FILE.')
C    ***********************************
C    *    SPECIFICATION FILE           *
C    ***********************************
         WRITE(1,210)
  210    FORMAT(/' NAME OF SPECIFICATION FILE ? ')
         READ(1,220) FNAME1
  220    FORMAT(4A4)
         WRITE(1,230)
  230    FORMAT(' NAME OF DATA FILE ? ')
         READ(1,220) FNAME2
         LENGTH = 13
         CALL  OPEN(6,FNAME1,LENGTH)
         READ(6,240) MAXOBS, NVARS
  240    FORMAT(I5,5X,I2)
```

```
            DO 260  IVAR = 1, NVARS
                READ(6,250)  (VNAMES(IVAR,J),J=1,NVARS)
      250       FORMAT(3A4)
      260   CONTINUE
            ENDFILE 6
            LENGTH = 2 + 4 * NVARS
            CALL  OPEN(7,FNAME2,LENGTH)
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     *     PROCESS                                             *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
            WRITE(1,310)
      310   FORMAT(/' GIVE RECORD NUMBER'
        2             /' OF RECORD TO DISPLAY.'
        3            //' USE RECORD NUMBER OF O'
        4             /' TO TERMINATE.')
      320   WRITE(1,330)
      330   FORMAT(/' RECORD NUMBER TO DISPLAY ? ')
            READ(1,340) IREC
      340   FORMAT(I5)
            IF  (IREC.EQ.0)  GO TO 400
            READ(7,REC=IREC)  NIREC,
        2        (VALUES(IVAR),IVAR=1,NVARS)
            IF  (NIREC.GT.0)  GO TO 360
            WRITE(1,350)
      350   FORMAT(/' RECORD IS EMPTY')
            GO TO 390
      360   DO 380  IVAR = 1, NVARS
                WRITE(1,370)  (VNAMES(IVAR,J),J=1,3),
        2            VALUES(IVAR)
      370       FORMAT(1X,3A4,F15.5)
      380   CONTINUE
      390   GO TO 320
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     *     END OF PROGRAM                                      *
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      400   ENDFILE 7
            WRITE(1,410)
      410   FORMAT(/' END OF PROGRAM' /)
            STOP
            END
```

TEST RUN

The following test run illustrates the use of the online inquiry program:

PROGRAM P0903

ONLINE INQUIRY SYSTEM
FOR DISK DATA FILE.

NAME OF SPECIFICATION FILE ? SPEC/DAT

NAME OF DATA FILE ? FILE/DAT

GIVE RECORD NUMBER
OF RECORD TO DISPLAY.

USE RECORD NUMBER OF 0
TO TERMINATE.

RECORD NUMBER TO DISPLAY ? 3

ERRORS	18.00000
AGE	6.00000
TIME	3.00000

RECORD NUMBER TO DISPLAY ? 0

END OF PROGRAM

9.4 Sequential Processing Economies

DISK FILE ORGANIZATION

The Radio Shack TRSDOS operating system for the Model III organizes the diskette into tracks, granules, sectors and records. Each sector contains 256 bytes. Each granule contains three sectors, and each track contains six granules. The five-inch floppy disk for the Model III contains 40 tracks.

Access to an adjacent track takes from three to 20 milliseconds depending on the disk controller system and make and model of disk drive. The diskettes spin at 300 revolutions per minute. This translates to five revolutions per second or 200 milliseconds per revolution. One millisecond is one thousandth of a second.

DISK ACCESS TIMES

The disk access time is the most important measure of perform-
ance. The computer will make no more than five disk accesses
per second. Online systems involve interactive data entry and file
updating. The random access capability of the disk drives is suf-
ficient from the standpoint of the user sitting at the keyboard if
the computer obtains the record on the first few attempts. This
is true of the statistical data base system. The record number gives
the relative position of the desired record. The system uses this
to calculate its exact location and obtains the desired record on
the first attempt.

SEARCHING

Searching or scanning a file can be very time-consuming. If the
search involves randomly jumping from location to location
within the file, the disk drives will not support more than five
accesses per second. The random access file system requires one
disk access per record during the search.

    The statistical data base system uses records of 14 bytes
each. Five disk accesses per second yielding 14 bytes per access
gives 70 bytes per second throughput. This is slow if the file
contains very many records.

BLOCKING

The standard block of data for floppy disk systems is 256 bytes.
Data on the diskette surface is organized into sectors of 256 bytes
each. The system automatically blocks sequential files with several
records per sector. The data is densely packed and automatically
spans from one sector to the following one.

    The system makes one access per sector. It does not make
one access per record. Some systems may even make one access
per granule or one access per track. The bigger the block of data
moved during the disk access, the greater the data throughput
for sequential processing. This also increases the amount of sys-
tem storage required for containing physical blocks of data.

    If the system accesses one 256-byte sector per disk access,
then the throughput will be about 1,250 bytes per second. If the

system accesses a three-sector granule at a time, the throughput will almost triple. It will fall short of this because a small amount of time is needed (11 milliseconds per sector) during the disk revolution to transmit the data.

SEQUENTIAL PROCESSING EFFICIENCY

If processing involves reading the file from beginning to end, sequential processing is usually faster. It may be more than 100 times faster than random access processing if the record size is very small. This is the reason why sequential processing methods retain their popularity.

There are two efficiency considerations. Sequential processing usually makes more efficient use of the computer, but often does not make efficient use of the people who use or update the data. Batch reports reflect what happened last month or last week but executives need current information. Efficient use of people suggests online systems with immediate access to information and immediate processing capability. There will be increasing use of computer terminals and online systems for this reason.

TRENDS

The choice between computer efficiency and personnel efficiency requires a compromise. Computer hardware costs are dropping rapidly, and personnel costs are rising rapidly. The inevitable result is a shift from emphasizing computer efficiency toward emphasizing personnel efficiency and overall performance of the system. The microcomputer is a prime mover in this shift.

SEQUENTIAL PROCESSING OF RANDOM FILES

Some systems including Microsoft FORTRAN for the Radio Shack TRS-80 provide efficient sequential access to random files. For those systems that have this capability, processing speeds and data throughput will approach those for true sequential files. This provides the benefits of both worlds. They provide high-speed sequential access of the entire relative file, and online inquiry and update in place of a random file. As microcomputers become more sophisticated, this capability will become more widespread.

PROGRAM LISTING CONTENTS OF A RANDOM FILE

The following program illustrates sequential access of the random file by printing the contents of the file:

```
          PROGRAM P0904
C     ***************************************
C     *     P0904                           *
C     ***************************************
C     AUTHOR
C          COPYRIGHT 1982
C          BY LAWRENCE MCNITT.
C     PURPOSE
C          PRINT CONTENTS OF
C          DISK DATA FILE.
C     SYSTEM
C          MICROSOFT FORTRAN
C          RADIO SHACK TRS-80.
C     ***************************************
C     *     ORGANIZATION                    *
C     ***************************************
C     INITIAL MESSAGE
C     SPECIFICATION FILE
C     PRINT HEADING
C     PRINT RECORDS.
C     END OF PROGRAM
C     ***************************************
C     *     VARIABLES                       *
C     ***************************************
C     VNAMES(64,3)  VARIABLE NAMES
C     VALUES(64)    VALUES FOR RECORD
C     FNAME1(4)     NAME OF SPECIFICATION FILE
C     FNAME2(4)     NAME OF DATA FILE
C     MAXOBS    MAXIMUM NUMBER OF OBSERVATIONS
C     NVARS     NUMBER OF VARIABLES
C     IVAR      CURRENT VARIABLE
C     IREC      CURRENT RECORD
C     NIREC     NEGATIVE OF CURRENT RECORD IF EMPTY
C     J         INDEX
C     LENGTH    LENGTH OF RECORD IN BYTES
C     ***************************************
C     *     INITIAL MESSAGE                 *
C     ***************************************
```

```
          DIMENSION VNAMES(64,3),VALUES(64),
     2                    FNAME1(4),FNAME2(4)
          WRITE(1,110)
110    FORMAT(/' PROGRAM P0904'
     2              //' PRINT CONTENTS OF'
     3               /' DISK DATA FILE.')
C      *********************************************
C      *     SPECIFICATION FILE                    *
C      *********************************************
          WRITE(1,210)
210    FORMAT(/' NAME OF SPECIFICATION FILE ? ')
          READ(1,220) FNAME1
220    FORMAT(4A4)
          WRITE(1,230)
230    FORMAT(' NAME OF DATA FILE ? ')
          READ(1,220) FNAME2
          LENGTH = 13
          CALL OPEN(6,FNAME1,LENGTH)
          READ(6,240) MAXOBS, NVARS
240    FORMAT(I5,5X,I2)
          DO 260 IVAR = 1, NVARS
              READ(6,250) (VNAMES(IVAR,J),J=1,NVARS)
250          FORMAT(3A4)
260    CONTINUE
          ENDFILE 6
          LENGTH = 2 + 4 * NVARS
          CALL OPEN(7,FNAME2,LENGTH)
C      *********************************************
C      *     PRINT HEADING                         *
C      *********************************************
          WRITE(2,310) FNAME1, FNAME2
310    FORMAT('1SPECIFICATION FILE ',4A4
     2            /' DATA FILE          ',4A4)
          WRITE(2,320)
320    FORMAT(/' VARIABLE NAMES')
          WRITE(2,330) ( (VNAMES(IVAR,J),J=1,3),
     2        IVAR=1,NVARS)
330    FORMAT(4(3X,3A4) )
C      *********************************************
C      *     PRINT RECORDS                         *
C      *********************************************
          DO 430 IREC = 1, MAXOBS
```

```
            READ(7,REC=IREC) NIREC,
     2             (VALUES(IVAR),IVAR=1,NVARS)
            IF (NIREC.LE.0) GO TO 430
            WRITE(2,410) IREC
 410        FORMAT(/' RECORD ',I5)
            WRITE(2,420) (VALUES(IVAR),IVAR=1,NVARS)
 420        FORMAT(4F15.5)
 430    CONTINUE
        WRITE(2,440)
 440    FORMAT(///' END OF DATA')
C   ***********************************
C   *   END OF PROGRAM                *
C   ***********************************
        ENDFILE 7
        WRITE(1,510)
 510    FORMAT(/' END OF PROGRAM' /)
        STOP
        END
```

TEST RUN

The following test run resulted from running the program:

```
PROGRAM P0904
LIST CONTENTS OF
DISK DATA FILE.
NAME OF SPECIFICATION FILE ? SPEC/DAT
NAME OF DATA FILE ? FILE/DAT
END OF PROGRAM
```

PRINTED OUTPUT

The following printed output resulted from the test run:

```
SPECIFICATION FILE SPEC/DAT
DATA FILE           FILE/DAT
VARIABLE NAMES
ERRORS          AGE           TIME
RECORD 1
      12.00000      2.00000        6.00000
```

RECORD 2
15.00000	4.00000	6.00000

RECORD 3
18.00000	5.00000	4.00000

RECORD 4
19.00000	3.00000	2.00000

RECORD 5
22.00000	6.00000	3.00000

END OF DATA

9.5 Exercises

1. Implement the programs of this chapter using random file techniques for the statistical data base.

2. Write a program that computes the average for each variable in the statistical data base.

3. Use the methods of this chapter to create and maintain a simple accounts receivable system. Each customer account should have an account balance, amount of purchases made during the month, amount of payments made during the month, and credit limit.

4. Write an online inquiry program for sales clerks to use to see if the current purchase will cause the current amount outstanding computed as

   balance + purchases – payments

   to exceed the credit limit. If the sale is authorized, add the sales amount to the purchases.

5. Write a customer payment program that allows the clerk to call up a customer account and add the amount of the payment to the amount of payments.

6. Write a summary program that updates the accounts receivable balance using the formula

   new balance = old balance + purchases – payments

   Clear the purchases field and the payments field to zero for the following accounting cycle.

Index

Petrocelli's "INVITATION TO" Series

Invitation to MAPPER:
A Pragmatic Approach to
End-User Computing–Book I

by Harry Katzan, Jr.

Invitation to FORTRAN for the TRS-80
Invitation to COBOL for the TRS-80

by Lawrence L. McNitt

Invitation to Ada & Ada Reference Manual
Invitation to FORTH
Invitation to PASCAL

by Harry Katzan, Jr.

Invitation to BASIC Fitness

by Stephen J. Mecca & Cemal Ekin