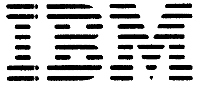


System/3 Model 6
Guide to BASIC
Learner-Paced Format

Student Text

System/3 Model 6
Guide to BASIC
Learner-Paced Format

Student Text

Preface

This Student Text directs the student through a self-study course in operating and writing programs for the IBM System/3 Model 6, using System/3 BASIC.

At selected points in the text are exercises directing the student to perform computer operations. Also in the text are progress checks called Practice in Programming. These progress checks ask the student to write a program to solve a given problem. Solutions are provided to indicate student understanding of the material studied.

The following publications should be kept available for reference:

System/3 BASIC Reference Manual (GC34-0001)
System/3 BASIC Operator's Guide (GC34-0003).

First Edition

Changes are continually made to the specifications herein; any such changes will be reported in subsequent revisions.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Education Development, 301 East Erie Street, Chicago, Illinois 60611.

© Copyright International Business Machines Corporation 1970, 1971.

Table of Contents

Section 1. Introduction 1

System/3 Components
System/3 BASIC Programming System

Section 2. Using System/3 BASIC 11

Preparing the System
Prior to Entering a Program
  LOGON
  EDIT
Entering a Program
Executing a Program
Saving a Program
  Editing a Saved Program
Signing Off

Section 3. Error Correction 21

Correcting Typing Errors
Correcting Syntax Errors
Non-Syntax Errors

Section 4. Input and Output Statements 27

Improving the Printed Result
  Character Variables
INPUT and GO TO Statements
  Terminating the Program
DATA and READ Statements
  Changing the Name of a Program
RESTORE Statement

Section 5. Assigning Values 37

The LET Statement
  Arithmetic Operators
  Levels of Priority
  Numeric Data
  Precision in Arithmetic Expressions
  System Constants

Section 6. Branching 43

GO TO. . .ON Statement
IF. . .THEN Statement
  Arithmetic Expressions in IF-expressions
  Character Expressions in IF-expressions
Writing Comments

Section 7. Looping 51

Program Loops
Loops in Series and Nested Loops

Section 8. Printing Results 59

The PRINT Statement
  PRINT Statement Without a Print List
  The Print List
  Commas as Separators

Controlling the Printhead Position
Continuing the Print-Out to the Next Line
Semicolons as Separators
Data Conversion
Rounding

Section 9. Formatting Results 69
The IMAGE and PRINT USING Statements
Using These Statements
Printing Decimal Values
Printing Numbers in the E-format
Printing Character Values
Multiple Use of IMAGE Statement
Printing Integers in Report Format
Printing Signs

Section 10. Functions and Subroutines 77
Functions
System Functions
User Functions
Subroutines

Section 11. Stopping Program Execution 85
STOP Statement
PAUSE Statement
SUSPEND and RESUME Commands

Section 12. Arrays and Matrices 89
Referencing Elements of Arrays
Defining the Shape and Size of Arrays
Creating Arithmetic Arrays
MAT READ Statement
MAT INPUT Statement

Section 13. File Processing 101
Creating Data Files
Using a Saved Data File
Saving a Program Associated with a File
Program - Generated Data Files
MAT GET Statement
MAT PUT Statement
Updating Files
RESET Statement

Section 14. Debugging 117
TRACE Command
STEP Command
DISPLAY and SET Commands
DISABLE and ENABLE Commands

Section 15. Program Modification Commands 125

Section 16. Libraries 129
The One-Star Library
LISTCAT Command
PROTECT Command
PULL Command
DELETE Command
The Two-Star Library

Appendix 135

Section 1 System/3 Model 6 Introduction

Performance Objectives:

Identify the System/3 Model 6 Components

Describe the relationship between the System/3 Model 6, System/3 BASIC, and the BASIC programming language

INTRODUCTION

BASIC is an easy-to-learn programming language similar to the everyday mathematical language of problem-solving. It is used with System/3 BASIC, a programming system to control the operations of the IBM System/3 Model 6, a compact yet powerful computer system for problem-solving.

SYSTEM/3 MODEL 6 COMPONENTS

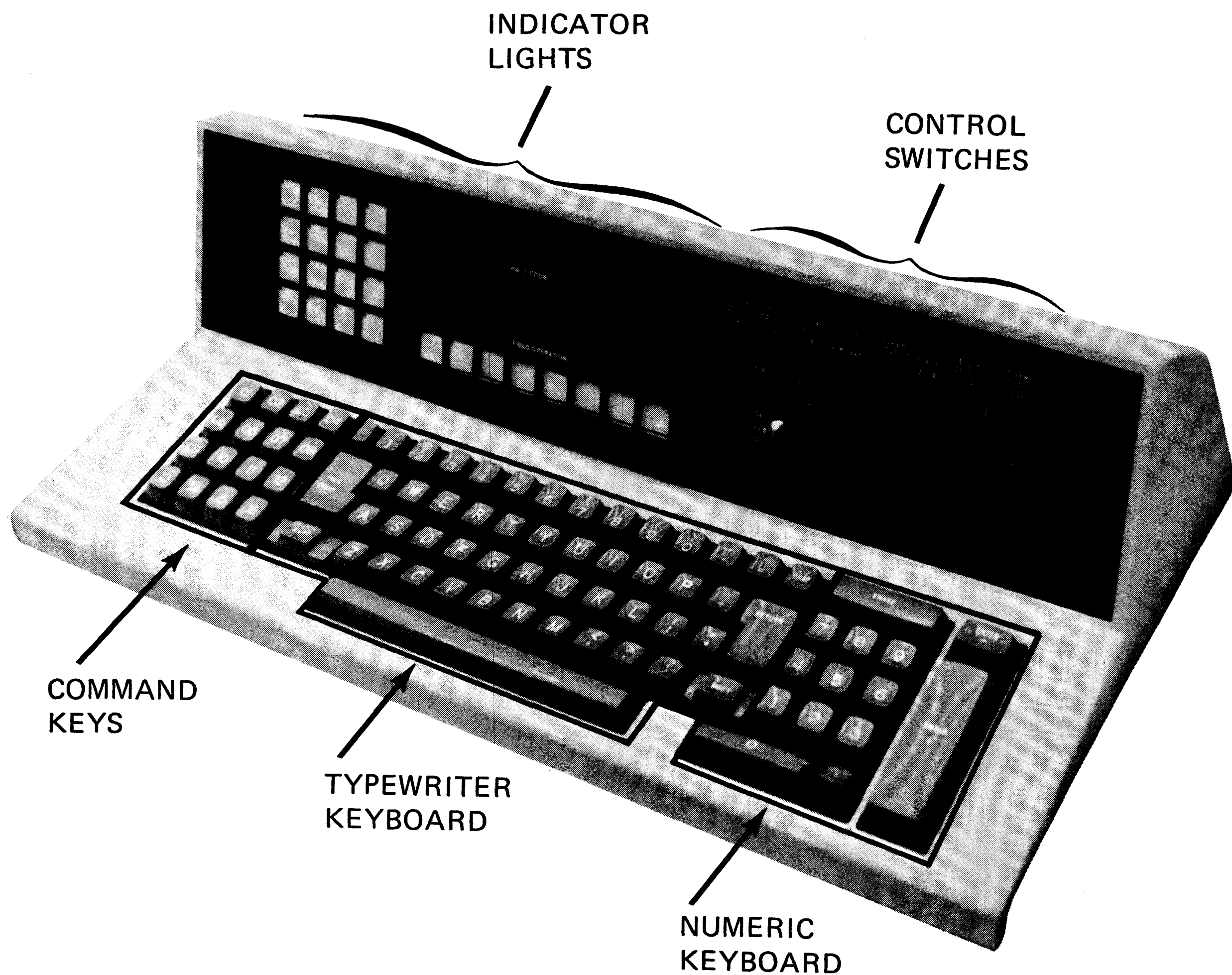
A computer system consists of input and output units, a processor, main storage, and auxiliary storage. On the System/3 Model 6, the combined components of the System/3 Model 6 occupy an area about the size of an office desk.

Console Keyboard

Information is entered through the Console Keyboard, the main input unit for the System/3 Model 6. Here are located all of the keys, switches, and indicator lights needed to communicate with the system. The keyboard portion of the Console Keyboard is divided into three sections:

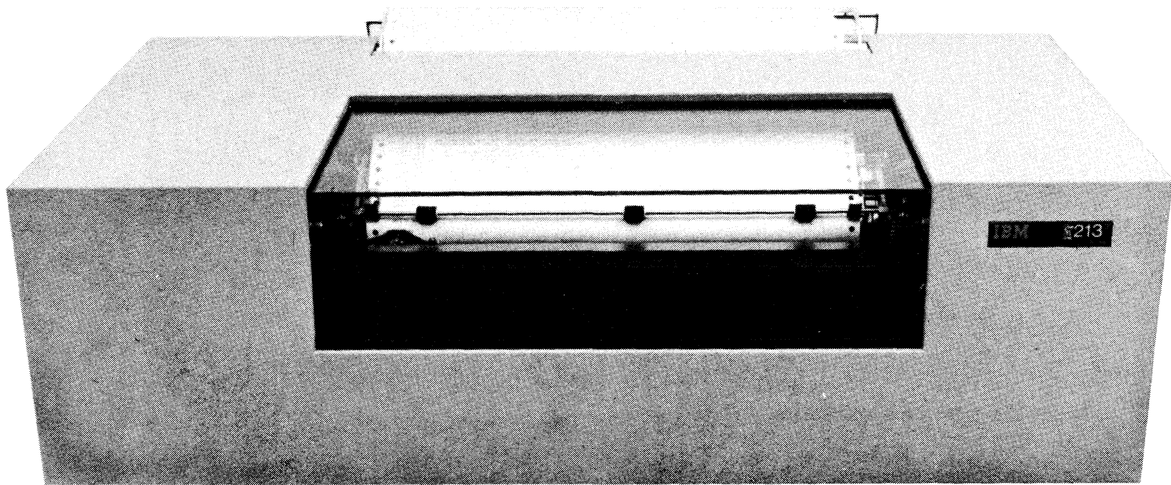
1. The center section resembles a typewriter keyboard. It has a 63-character alphanumeric keyboard with standard typewriter function keys: shift, backspace, tab, and return. It also has a special function key, PROG START (program start).
2. The section on the right has a key arrangement that is similar to an adding machine. In addition to the 10 numeric digit keys, it has the ENTER +, ENTER -, "." (decimal point), and ERASE keys.
3. On the left are the command keys. Those labeled 01 through 08 are standard. (Command keys 09 through 16 are needed if a Cathode Ray Tube Display Station - CRT - is attached to the system).

The console portion is divided into two sections. Switches necessary for operator control of the system are on the right, and program indicator lights are on the left. The functions of these keys and switches will be described later in this text.



IBM 5213 Printer

A line printer is the primary System/3 Model 6 output device. Positioned above and behind the keyboard, it prints out processing results and messages from the system. It also prints the information entered from the keyboard. The standard printer has a 13" typewriter-like carriage. It prints 132 characters per line at a speed of 85 characters per second. A 22" carriage printer (220 characters per line) and a faster bi-directional printer are optional devices available.

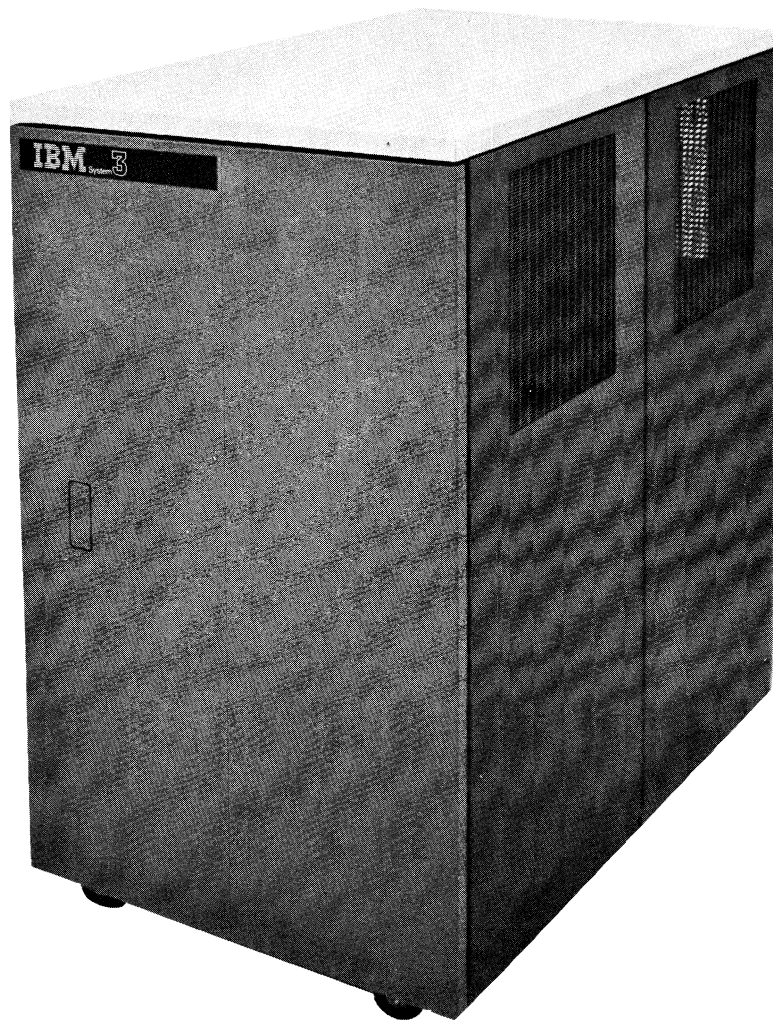


IBM 5406 Processing Unit

The processing unit at the right of the keyboard is the control center of the system. It is made up of three parts: a control section, an arithmetic and logic section, and main storage.

The *control section* coordinates all the functions of the computer. It reads and interprets stored programs and then directs the other sections and units of the system to execute the operations specified in the program. A program is a sequence of instructions given to the computer to execute in order to produce a desired result.

Main storage is somewhat like a series of numbered mailboxes, each of which is a storage location capable of holding data and program instructions. Main storage is available in several capacities, starting with 8K, where K stands for 1024 storage locations. With 8K of storage, 8192 (8 x 1024) characters of information can be stored. Additional storage capacities of 12K or 16K are also available. Execution of arithmetic functions such as adding, subtracting, multiplying, etc., and logical operations, such as comparing, are performed in the *arithmetic and logic section* of main storage.



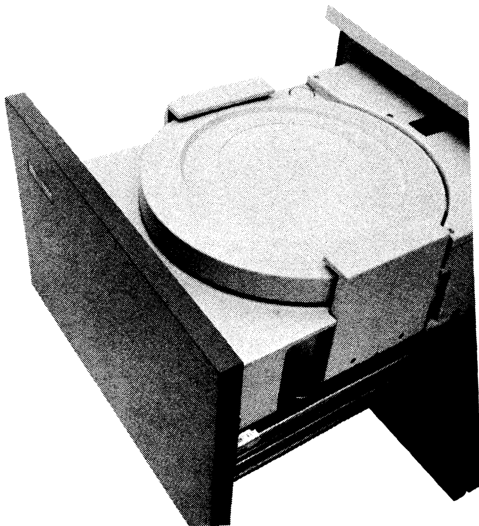
IBM 5444 Disk Storage Drive and 5440 Disk Cartridge

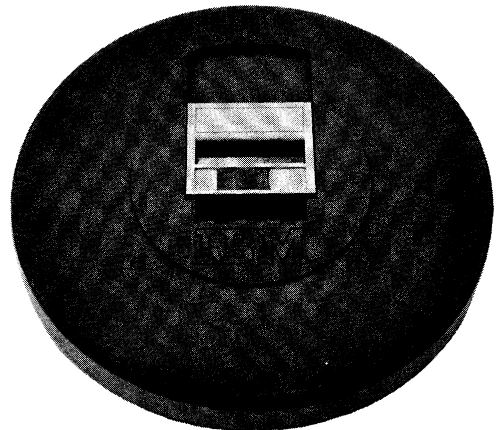
Magnetic disks provide auxiliary storage. Since main storage does not have the capacity to hold all data and programs, only the information in immediate use by the computer - the instructions being executed, the data being processed - is in main storage. All other information is stored on disk. The programs you enter through the keyboard go to disk to await execution. Programs that have been executed but which you need for future use are also saved and stored on disk.

A disk is a round metal plate, coated on both sides with a magnetic material on which information is electronically recorded. Like main storage, numeric, alphabetic, and special characters are recorded on the disk. The disk is mounted on a drive - a revolving shaft that spins the disk like a phonograph record. Disks are housed in two sliding drawers on the lower left side of the system. The top drawer is labeled Disk Drive 1; the bottom drawer, which is optional, is Disk Drive 2.

Two disks, one fixed and one removable, are mounted on each disk drive attached to the system. The fixed disks, called F1 and F2, cannot be removed. The removable disks, called R1 and R2, can be replaced by other disk cartridges allowing practically unlimited auxiliary disk storage.

When the system is in operation, a magnetic read/write head moves across the disks, reading information from disk into main storage or writing on disk information sent from the Processing Unit.





Minimum System Requirements

Certain optional units can also be part of the system and can also be used with System/3 BASIC. One of these is the IBM 2265 Display Station, a cathode ray tube unit which, like a television set, shows the output on a screen. Another is the IBM 5496 Data Recorder, an input/output unit that reads and punches 96-column cards. Although these and other optional devices mentioned might be part of your computer system, they are not necessary to study this course in BASIC. Only the minimum components for the System/3 Model 6 are required to use System/3 BASIC:

Central Processing Unit 8K main storage
Console Keyboard with 8 command keys
5213 Model 1 Matrix Printer (132 print positions)
5444 Model 1 Disk Storage Drive
5440 Disk Cartridge

BASIC PROGRAMMING LANGUAGE

The BASIC programming language consists of approximately 30 statements that you can use to communicate with the computer system. Each statement is an instruction to the computer to perform some specific task. The LET statement, for instance, instructs the computer to assign a value to a variable:

```
LET A = (4000 + 200) / 7
```

The expression on the right of the = sign is computed before it is assigned to the variable A. In a PRINT statement you describe the output you want:

```
PRINT A
```

When this PRINT statement is executed, the value assigned to A (600) will be printed. An END statement tells the computer there are no more instructions:

```
END
```

A group of statements that are arranged in logical sequence to solve a problem forms a program:

```
100 LET A = (4000 + 200) / 7
110 PRINT A
120 END
```

Program statements are entered through the Keyboard or the Data Recorder and then stored on disk to await execution.

SYSTEM/3 BASIC PROGRAMMING SYSTEM

Before instructions can be executed, the BASIC statements must be converted into the language of the computer. Converting BASIC statements into machine language is the job of a special program called the compiler. In addition, the programs and data you enter must be checked and controlled as they move through the various units of the computer system.

For this, control programs are required. Finally, in order to save, modify, and reuse programs and data, utility programs are needed. The compiler, control programs, and utility programs make up the programming system called System/3 BASIC. These programs are already written and are available from IBM.

System/3 BASIC is stored on a disk that is mounted on Disk Drive 1. These system programs are controlled through switches on the Console and through system commands entered via the Keyboard or the Data Recorder. Before any computer operations can be performed, a control program called the Supervisor must be loaded from the disk into main storage. (The procedure for loading the Supervisor - IPL procedure - is explained in Section 2.)

To specify the operation to be performed, an appropriate command is entered. For example, after program statements are entered, the RUN command is keyed. The RUN command orders the system to compile (convert to machine code) and execute the program:

```
100 LET A = (4000 + 200) / 7   }  BASIC statements
110 PRINT A                    }  entered through Keyboard
120 END                        }  and waiting on disk
RUN                            }  Execution command
                                  entered through Keyboard
```

The program is compiled and executed in main storage. The Supervisor orders the compiler and necessary control programs from the disk into main storage as they are required to perform operations. During system operation, information is in a continual interchange between disk and main storage.

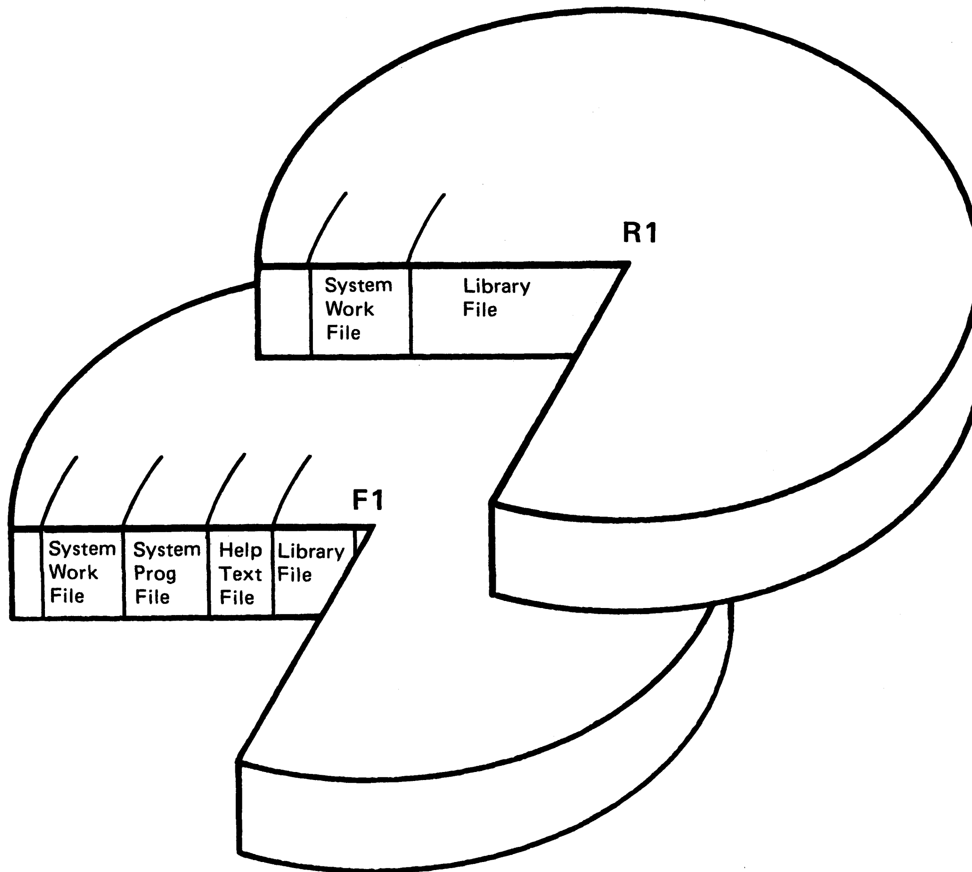
System Generation

The set of programs that make up the System/3 BASIC is provided by the IBM Program Information Department on a user-supplied disk cartridge, usually referred to as the PID cartridge. In most cases, the PID cartridge is not used during normal computer operations. Instead, a disk containing copies of the system programs is mounted on a disk drive. During a special procedure called *system generation*, the System/3 BASIC programming system is copied from the PID cartridge to another disk. This disk, then, is a duplicate of the PID cartridge; the original PID cartridge is kept as a back-up system.

The disk that contains the programming system is organized into specifically defined areas. One of these is the *System Program Area.* This is where the system programs - compiler, control programs, utility programs, etc., are stored. Another area contains the HELP Text, a file of reference information about the System/3 BASIC that the system will print out on request to assist in diagnosing programming errors.

To store data and programs there is a *File Library Area.* Information saved in this area can later be reused. A sample program of the type that you can write is included in the File Library Area on the PID cartridge.

A System Work Area is also required. This is the area where the programs you enter through the Keyboard or Data Recorder are temporarily stored. From here, your program can be brought into main storage for execution, or it can be copied to the File Library Area and saved for future use. (Since a System Work Area does not exist on the PID cartridge, it cannot be copied and must be assigned on the disk that contains the programming system. A command is entered to assign this space.)



The figure above shows these areas organized on two disks, F1 and R1. The System Program Area, the HELP Text File, a File Library Area, and a System Work Area are on F1. Additional File Library space and a System Work Area are located on R1. This is not the only possible arrangement, but there are two rules that must be observed in organizing disks:

1. The System Program Area must be on either R1 or F1.
2. A System Work Area must exist on both R1 and F1.

Although additional File Library space is not a requirement for using the system, it is usually assigned on a second disk. This additional space becomes necessary as the number of programs you want to save increases.

The system generation procedure is performed when the computer system is first installed, and unless you are responsible for the installation, you will not have to concern yourself with the procedure. However, you will have to know:

1. The disk (R1 or F1) on which the System Programs are located
2. Which disk (R1, F1, R2, F2) has a File Library Area in which you can save your data and programs.

If System Generation has not been performed, refer to the Appendix for an explanation of the procedure. Otherwise, proceed to Section 2.

Section 2 Using System/3 BASIC

Performance Objectives:

- Prepare System/3 Model 6 for System/3 BASIC operation
- Enter and execute programs
- Modify and list programs
- Save programs

PREPARING THE SYSTEM

To prepare the computer for operations, perform the following six-step procedure.

Step 1: If POWER is OFF, move the POWER switch to ON.

Step 2: Mount the BASIC System Program disk, if it is not already on a disk drive. (Disk Drive switches must be OFF).

Step 3: Set Disk Drive 1 (and 2) switch to ON. Wait about 60 seconds.

Step 4: To start system operation, perform the following Initial Program Load (IPL) procedure.

A) Move the DISK SELECT switch to FIXED or REMOVABLE, whichever contains the BASIC System Programs.

B) Lift the PROGRAM LOAD switch. Because this switch is spring-loaded, it returns automatically to its former position. Turning on this switch loads the control program called the Supervisor into main storage. The system responds by printing:

ENTER CONFIGURE COMMAND OR PRESS PROG START KEY

Note: The option to "enter CONFIGURE command" refers to specifying the components available on the computer. During system generation, a list of available components was entered and a configuration record was written on Disk F1. This configuration record should be modified only if the machine configuration changes (e.g., an 8K system is increased to 12K or a Data Recorder becomes part of the system).

C) On the keyboard, press the PROGRAM START key.
The system prints:

ENTER DATE MM/DD/YY

D) Key in the current date: month/day/year. Slashes must be entered.

EXAMPLE: 01/06/71

If you make an error in keying, press the ERASE key and key the date again. When the date is entered correctly press the RETURN key. If the system responds with a question mark or again prints ENTER DATE MM/DD/YY, the date was not keyed in the proper format. Rekey the date.

Step 5: If the system prints any error messages (disk not initialized, no work area assigned), correct the error conditions. The procedures for initializing a disk and assigning a work area are described in the appendix.

Step 6: If your system has a CRT, turn the CRT power switch on (lower right front of the CRT) and then enter the WRITE command in the format shown below:

WRITE PRINTER, CRT then press the RETURN key
or
Press Command Key 10

Both the Printer and the CRT will function as output units.

Note: Throughout this text are exercises that require you to use the system. Each time you begin the exercises make sure the computer has been prepared for operations.

PRIOR TO ENTERING A PROGRAM

Before entering a program, you key two commands to the system: the LOGON command and the EDIT command.

LOGON Command

The LOGON command puts a password into effect. The *password* is the name that identifies all the saved programs that belong to you. Since you will save the program that you are going to enter, it is important that the LOGON command is keyed so that the program will be associated with your password.

All programs saved for future use are stored on disk in the File Library Area. To guarantee security for everyone, each user has a password which, like a key, provides access to his saved programs. Any programs saved when the password is in effect can be used later only if the same password is in effect.

The following information is given with the LOGON command:

User Password
Disk Label
Password Status

Password

The *password* can be any name you decide to use. There are only two restrictions:

1. The password must begin with an alphabetic character.
2. The password can be no longer than eight characters (alphabetic and/or numeric) in length.

EXAMPLE: LOGON SMITH
LOGON USER2
LOGON TR

Disk - Label

The *disk - label* is the name assigned to a disk when it is initialized. (Initialization is a procedure performed the first time a new disk is used on the system. This procedure is explained in the Appendix.) By specifying the disk label with the LOGON command, you indicate to the system on which disk programs are to be saved. If the disk - label is not included when you LOGON, the programs saved will be stored in the File Library Area on R1.

EXAMPLE: LOGON SMITH/ DISK4 *The disk with the label DISK4 will be used to save programs associated with the password SMITH.*

LOGON USER2 *No disk specified so any programs saved will be stored on R1.*

Password Status

NEW must be specified the first time your password is used with a particular disk. This places your password in the password directory contained on the disk.

EXAMPLE: LOGON SMITH/DISK4, NEW

After the command has been keyed, press the RETURN key. The system responds by printing:

READY

EDIT Command

The EDIT command is used to name the program you are going to enter. This command clears the Work File, that location within the System Work Area reserved for temporary storage of programs, and makes it available for the program specified in the EDIT command. The command is entered in the following format:

EDIT filename

The *filename* can be any name you want to give your program. This name, however, must be different from the names of any programs already in your library. Like the password, it must begin with an alphabetic character and must be eight characters or less (alphabetic and/or numeric) in length.

EXAMPLE: EDIT STOCK24

If the command is entered in the proper format, pressing the return key causes the system to print:

WORK FILE HAS BEEN CLEARED AND NAMED filename
READY

Error Correction

Should the system respond with a question mark or an upward arrow instead of the message or the word READY, then the command was not keyed in the proper format. Simply rekey the command in the proper format and press the RETURN key. If you key part of the command and make an error, press the ERASE key and rekey the line, or backspace to the point of error, key the correct character and the remainder of that line.

EXERCISE 2 - 1:

1. Decide on a name to use as a password. If you know the disk-label of the disk you will use to save your programs, enter the LOGON command in the following format: LOGON password/disk-label, NEW

   If you don't know the disk-label and prefer to use the File Library Area on R1, use the format below:

   LOGON password, NEW

2. Enter the EDIT command in the correct format, i.e., EDIT filename. Use the filename STOCK for the program you are going to enter.

ENTERING A PROGRAM

Once the Work File has been cleared and named, you can enter a program to solve a problem. The problem described below relates to the stock market.

Given: Stock price (P) = 18.75

Annual dividend (D) = 1.05

Find: % return on investment or yield (Y)

To solve, multiply the annual dividend by 100 and divide by the stock price:

$$Y = \frac{D \times 100}{P}$$

The solution is described similarly with BASIC statements.

BASIC Statements

The set of statements that describe the solution to the problem make up a *program*. To solve the STOCK problem three types of statements must be used. The first three statements of the program are LET statements:

```
100 LET P = 18.75
110 LET D = 1.05
120 LET Y = D * 100 / P
```

In this example, the value of Y is determined by multiplying the value of D by 100 and dividing by the value of P. The symbols * and / represent multiply and divide functions.

A LET statement assigns a value to a variable. In the STOCK program, P, D, and Y are names for arithmetic variables. To P the LET statement assigns the value 18.75; to D the value 1.05. The variable Y takes on the value of the arithmetic expression D*100/P, which will be computed before it is assigned to Y. In BASIC, the name of an arithmetic variable consists of a single letter or a letter followed by a number, 1 through 9.

EXAMPLE: A
B1
C2
D

The PRINT statement, which is the next statement in the program, describes the desired output.

```
130 PRINT Y
```

When this statement is executed, the value assigned to Y is printed.

The last statement in the program, the END statement, indicates that no more program statements follow.

```
140 END
```

The program ends when this statement is executed. There can be only one END statement in a program.

BASIC Statement Lines

Each BASIC statement must be contained within a single line. A statement cannot be continued to the next line. To get the desired result, statements must be executed in proper sequence. The sequence of execution is determined by the line number. For this reason, each BASIC statement line must begin with a line number.

100	LET P = 18.75
Line number	BASIC statement

A line number can be any number from 0 to 9999. In the STOCK program, lines are numbered in increments of 10, beginning with 100. Using increments of 10 allows you to insert additional statements at a later time.

The Input Status

A BASIC program can be entered into the Work File only when the system is in the *input status.* The system indicates this status by printing READY. The program is entered line-by-line, first the line number and then the statement. At the end of each line the RETURN key is pressed.

EXECUTING A PROGRAM

BASIC statements entered through the keyboard are stored on disk in the Work File. The program will be executed when you enter the RUN command.

When the RUN command is entered, the statements stored in the Work File are compiled (converted to machine language) and then executed. The result is printed when the PRINT statement is executed. Execution stops when the END statement is encountered; the word READY is then printed to indicate that the system has returned to the input status.

Even after execution, a program remains in the Work File in statement line form until the Work File is cleared. Consequently, the program can be executed again by re-entering the RUN command.

To Enter Commands

Like BASIC statements, commands are entered in lines and the RETURN key is pressed to complete each entry. Unlike statements, however, command lines do not have a line number and a space is required after the command keyword. The keyword is the first word of the command.

EXAMPLE: EDIT STOCK

Also, command lines are not stored in the Work File but cause immediate system action. Several of the frequently used commands may be entered by pressing a command key. For instance, instead of keying RUN, you can enter the RUN command by pressing Command Key 08.

EXERCISE 2 - 2

1. Enter and execute the STOCK program. (Numbers may be entered through either the numeric keyboard at your right, or the typewriter keyboard in the center.)

   If you make a typing error before the RETURN key is pressed, use the ERASE key and rekey the entire line, or backspace to the point of error and rekey the line from the point of error. If you notice the error after the RETURN key is pressed, or if the system prints an arrow or a question mark after the line is entered, rekey the line correctly.

   ```
   100 LET  P = 18.75
   110 LET  D = 1.05
   120 LET  Y = D * 100 / P
   130 PRINT  Y
   140 END
   ```
   RUN (This command can be keyed or entered by pressing Command key 08.)
2. Rerun the STOCK program.

Changing Values in the Program

The values in the STOCK program can be changed by clearing the Work File and then entering the same program but with the new values. To clear the Work File, the EDIT command is entered with the filename of the program.

Certain features on the system can save time in entering the program. For instance, line numbers can be obtained automatically by pressing the PROG START (program start) key at the beginning of a line. When the PROG START key is pressed at the left margin, the system generates a line number that is 10 greater than the last line number entered. If the PROG START key is pressed before the first line of the program is entered, the system will generate the number 100 as the first line number of the program.

You can also enter the program more quickly if you omit the word LET in the LET statement. Since the word LET is optional, the statement can be keyed in either of two ways:

```
100 LET P = 18.75
     or
100 P = 18.75
```

Another option is the use of blanks. Blanks may be used freely within a BASIC program statement line. For example, Line 120 can be entered in any of the following ways:

```
120 Y = D * 100 / P
120  Y    =   D  *  100    /P
     12     0    Y = D * 100/     P
120 Y = D      *     100  /  P
```

EXERCISE 2 - 3

1. Clear the Work File and name it STOCK.
2. Enter the STOCK program shown below. Use PROG START at the beginning of each line to generate the line number. Omit the word LET in the LET statements.

   ```
   100 LET P = 278.50
   110 LET D = 4.60
   120 LET Y = D * 100/P
   130 PRINT Y
   140 END
   ```

3. RUN

   Your result should be 1.65171

Rather than clearing the Work File and entering the entire program, values can be changed by simply changing the appropriate lines in the Work File. The rest of the program statements remain unchanged. To provide new values for the arithmetic variables P and D in the STOCK program, lines 100 and 110 should be changed. If the system is in the input status (READY), simply key in the line number of the statement that is to be changed. Then key the statement in the desired form and press the RETURN key. In the Work File, the newly entered line replaces the line with the same number.

In changing the STOCK program lines, do not use the PROG START key to enter line number 100. By pressing the PROG START key, a number 10 greater than the last line number entered is generated. Since 140, the last line of the program, is the last line number entered, pressing the PROG START key would cause 150 (140 + 10) to be printed. After lines 100 and 110 have been changed, you enter the RUN command. The system will compile and execute the program with the new values.

Listing the Program

The LIST command provides a program listing in line-number sequence regardless of the original statement entry order. Command Key 05 can be used to enter this command in lieu of keying LIST.

EXERCISE 2 - 4

1. LIST the program in the Work File
2. Change statement lines 100 and 110 of the STOCK program to the following:

   ```
   100 P = 77.25
   110 D = 4.20
   ```

3. LIST and then RUN. Note that the system automatically puts the statements in line-number sequence regardless of the sequence in which they were entered.
4. Change the value of P to 47.11 and the value of D to 3.10. LIST and RUN.
5. Change P to 82.75 and D to 3.50. RUN

SAVING A PROGRAM

If the program has executed satisfactorily and you would like to use it again at some future time, you can place it in the File Library Area. To do so, you enter the SAVE command by keying:

SAVE

Command key 03 can also be used to enter this command. When the SAVE command is entered, whatever is in the Work File is copied to the File Library Area of the disk specified in the LOGON command. If a disk was not specified in the LOGON command, then the Work File is copied to the File Library Area on R1.

The program is saved in the File Library Area under the name with which it was entered, that is, the filename used with the EDIT command.

A SAVE command does not affect the Work File. A program copied from the Work File to a File Library Area remains in the Work File until cleared.

EXERCISE 2 - 5

1. LIST the program now in the Work File.
2. SAVE this program. Command key 03 can be used to enter the SAVE command.
3. Once again LIST the Work File. Note that the program copied to the File Library Area is still in the Work File.
4. Key EDIT TEST to clear the Work File.
5. To verify that the Work File is empty, key LIST.

Editing a Saved Program

A program saved in a File Library Area is associated with the password that was in effect when SAVE was entered. Before the saved program can be used, this same password must be in effect.

To use a saved program, you must first copy it into the Work File. This is done by entering the EDIT command with the name of the program.

EXAMPLE: EDIT STOCK

When this command is entered, the saved program exists in two places: The File Library Area and the Work File.

EXERCISE 2 - 6

1. Copy the saved STOCK program into the Work File
2. LIST the Work File
3. RUN the STOCK program

SIGNING OFF

Whenever you finish your work at the computer, key the OFF command. This cancels your password and prevents the next user from accidentally using your library. It also clears the Work File.

If the printer was being used, the paper automatically spaces up so that you can tear off your work. If the CRT is part of the system and was being used, the display screen is cleared.

To put your password back into effect, you enter the LOGON command. Since your password is in the disk password directory, NEW does not have to be specified.

EXERCISE 2 - 7

1. Key OFF
2. LOGON if you are continuing with Section 3 of this text.

Section 3 Error Correction

Performance Objectives:

- Correct typing errors
- Correct syntax errors
- Identify non-syntax errors

CORRECTING TYPING ERRORS

A typing error that is detected before the RETURN key is pressed can be corrected by pressing the ERASE key and then rekeying the line. Or the BKSP (backspace) key can be used. The BKSP key moves the paper up one line and also moves the print head one position to the left. If held down, backspacing continues until the key is released. The BKSP key is used to move the print head back to the position where the error occurred. Correct characters can then be keyed to replace those that are wrong.

A typing error detected after the RETURN key is pressed is corrected by rekeying the line. Some additional system features allow this line to be corrected without having to rekey those parts of the line that are correct. These are the ENTER - (enter minus) key and the TAB key.

The ENTER - key causes the printer to space one line without changing the print head position. It permits reading the printed line suspected of containing a typing error.

The TAB key duplicates characters from the previous line. It allows the copying of correct characters from the line containing an error. When held down, tabulating continues until the key is released.

After an error correction procedure in which only part of a line is rekeyed, the printed line looks incomplete. One part of the line may be two or three print lines above the other part of the line. In the Work File, however, the line is stored as one complete and correct line. The LIST command followed by the line number of the corrected line can be used for a print-out of the statement as stored in the Work File.

EXAMPLE: LIST 110

EXERCISE 3 - 1:

1. Clear the Work File and name it ERRORS. Then enter the following line but <u>do not</u> press the RETURN key:

   100 Y = D * 100 / P

   Change Y to Z with the following error correction procedure:

   a. Press the ENTER - key once or twice
   b. BKSP to Y
   c. Key Z
   d. TAB over the correct characters in the line
   e. Press the RETURN key

2. LIST the contents of the Work File

3. Enter the following line and press the RETURN key:

   110 R = A + C - 16

   Change C to B with the following procedure:

   a. TAB to C
   b. Replace C with B
   c. TAB to end of statement
   d. Press RETURN

4. LIST 110 (Only line 110 is listed.)
5. Delete line 100 from the Work File as follows:
   a. Key 100
   b. Press RETURN
6. LIST the contents of the Work File. Line 100 has been deleted.
7. Delete line 110
8. LIST the contents of the Work File. The Work File is empty.

CORRECTING SYNTAX ERRORS

A syntax error is a violation of the rules for writing statement and command lines. The system detects syntax errors when the RETURN key is pressed at the end of a line. It indicates an error by printing an upward arrow under the *first* item in error. For instance, if you forget to enter the * to indicate multiplication, the system prints an arrow under the position of the second multiplication factor.

```
EXAMPLE: 0170 A = B + 7 (C + D)
                        ↑
```

A line that contains a syntax error is rejected by the system. A command with a syntax error is not executed, and a statement with a syntax error is not entered into the Work File.

In most cases you will know why the syntax error occurred. If not, the system can provide more detailed information. Press the ENTER + key on the numeric portion of the keyboard, and the system will print a text giving the reason for the error. If the ENTER + key had been pressed after the error shown above had been detected, the system would print the following message:

ERROR 019 ⟨OPERATOR⟩ REQUIRED BTWN LAST 2 CHARACTERS CHECKED

Still more information can be obtained by using the HELP TEXT provided with the system. This text is a file of reference information stored on disk. Through use of the HELP command, information about a specific topic can be printed.

If you enter only the word HELP, the system prints a list of topics from which you may select. But you can get information about a particular topic more quickly if you use the keyword that appears in the error message which is printed when the ENTER + key is pressed.

In the error message, terms that are enclosed in angle brackets are *keywords* that can be used with the HELP command.

EXAMPLE:

ERROR 019 ⟨OPERATOR⟩ REQUIRED BTWN LAST 2 CHARACTERS CHECKED

To learn more about operators, for example, key

HELP 'OPERATOR'

The angle brackets in the error message are replaced by single quotation marks when the keyword is used with the HELP command.

The system is not able to diagnose all syntax errors, and prints a question mark when this kind of error is detected. For instance, if you wanted to turn off the system and mistakenly entered a non-existent LOGOFF command (instead of the OFF command), the system would respond with a question mark.

User: LOGOFF
System: ?

In this case no error message will be printed if the ENTER + key is pressed. The line must be rekeyed.

A line with a syntax error can be corrected by rekeying the line. However, corrections can sometimes be made more quickly if you use Command Key 04.

After the arrow indicating the syntax error appears, press Command Key 04. All characters of the last line preceding the up-arrow are printed. You then correct the error and complete the line.

EXERCISE 3 - 2:

1. Enter the following incorrect line:

   100 A = B + 7 (C + D)

   a. Press RETURN key
   b. Press the ENTER + key
   c. Enter HELP 'OPERATOR'. When a list of topics is printed, select A by keying A and pressing the RETURN key.
   d. Enter the error line again and correct using Command Key 04, rekeying the line from the point of error.

2. Enter the following line with PRINT spelled incorrectly:

   110 PRNT Y

   a. Press RETURN
   b. Press ENTER +
   c. Enter HELP 'PRIMARY KEYWORD'
   d. Enter the incorrect line again and correct using Command Key 04

NON - SYNTAX ERRORS

The system detects non-syntax errors during compilation or execution of a program or command. When the error is detected, the system prints a message indicating the reason for the error. For example, if you try to enter a new password on a disk that already contains that password, the system rejects the password and prints the message:

ERROR 380 ⟨PASSWORD⟩ PREVIOUSLY DEFINED

When an error occurs within a program, the system prints the line number as well as the reason for the error. For instance, if an attempt to divide by 0 is made in line 130 of a program, the system prints the following message:

ERROR 790 AT LINE 130 DIVISION BY ZERO

The error number indicated in the message can be used as a reference to the *System/3 BASIC Operator's Guide* (GC34-0003). The Operator's Guide documents each error, giving the cause, the system action, and the recovery procedure. If the error message contains a HELP keyword (indicated by angle brackets), you can enter the HELP command with the keyword for additional information.

Sometimes a program executes without an error being detected but does not produce the result you want. To correct an error like this, it is necessary to analyze your approach to the problem. System/3 BASIC provides tools to help you (TRACE Command, PAUSE statement, etc.). These will be discussed later in the section "Debugging".

EXERCISE 3 - 3:

1. Clear the Work File and name it ERRORS
   a. Enter the LIST command
   b. Use the keyword in this non-syntax error message with the HELP command.
2. Enter and execute the following program which will cause an execution error:

   ```
   100 R = 50
   110 X = R / S
   120 S = 25
   130 PRINT X
   140 END
   RUN
   ```

   This is an example of a programming logic error. The HELP command will not assist you in determining how to correct this type of error since no rules of BASIC have been violated.
3. The error, division by zero, results from the division operation (R/S) occurring before a value is assigned to S. To correct, the program statements must be resequenced, i.e., the statement that assigns a value to S must precede the statement that assigns a value to X.

   Make the correction and RUN. The answer is 2.

Section 4 Input and Output Statements

Performance Objectives:

- Print character constants
- Enter data with INPUT statements
- Enter data with the DATA and READ statements
- Use RESTORE statements

IMPROVING THE PRINTED RESULT

The printed result of the STOCK program would be more meaningful if it were identified on the print out:

```
STOCKPRICE =  77.25   DIVIDEND = 4.20   YIELD = 5.32 PERCENT
```

In the PRINT statement that causes this line to be printed, two kinds of data are used: arithmetic variables and character constants. This statement would appear:

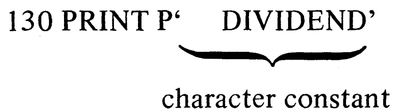
```
130 PRINT 'STOCKPRICE=' P  'DIVIDEND=' D  'YIELD=' Y  'PERCENT'
```

The arithmetic variables are P, D, and Y. The system prints the values assigned to these variables.

The character constants are 'STOCKPRICE=', 'DIVIDEND=', 'YIELD=', and 'PERCENT'. The system prints this information exactly as it appears in the PRINT statement but without the quotation marks. A character constant in a PRINT statement may consist of any number of printable characters, including blanks, numbers, and special characters, and must always be enclosed by single quotation marks.

Spacing between a printed character constant and a variable value is achieved by inserting blanks at the beginning or the end of the character constant in the PRINT statement. For example, to have 5 spaces inserted between the value of P and the character constant DIVIDEND, enter 5 blanks before the word Dividend.

```
130 PRINT P'     DIVIDEND'
```



character constant

Blanks outside of the quotation marks have no effect on the printed result. For instance, no spaces occur between the value for P and DIVIDEND if entered as follows:

```
130 PRINT   P   'DIVIDEND'
```

To achieve spacing in the printed result of the STOCK program, the PRINT statement could be entered as follows:

```
130 PRINT 'STOCKPRICE=' P  '    DIVIDEND=' D  '    YIELD=' Y  '  PERCENT'
```

EXERCISE 4 - 1

1. EDIT the STOCK program (which was previously saved).

Note: To retrieve the STOCK program from the library, the password in use must be the same password that was in effect when this program was saved.

2. LIST the STOCK program.

3. RUN
4. Change the PRINT statement so that the values for P, D, and Y are printed along with the character constants 'STOCKPRICE=' 'DIVIDEND=' 'YIELD=' and 'PERCENT'. Use whatever spacing you consider appropriate between the variables and the character constants.
5. RUN

Character Variables

Suppose you wanted the name of a company, for instance BIG OIL, to appear in the printed result:

```
BIG OIL STOCKPRICE=77.25    DIVIDEND=4.20    YIELD=5.32  PERCENT
```

BIG OIL could be included as a character constant in the PRINT statement:

```
130 PRINT 'BIG OIL   STOCKPRICE=' P '    DIVIDEND=' D '    YIELD=' Y '   PERCENT'
```

But this would mean that everytime a different stock was analyzed, a new PRINT statement would have to be entered. This can be avoided by using a character variable.

A character variable, like an arithmetic variable, can be assigned different values. The name of a character variable consists of an alphabetic character followed by a dollar sign.

```
EXAMPLE:   A$
           B$
           Z$
```

Only a character value can be assigned to a character variable. The assignment can be made with a LET statement. By adding another LET statement to the STOCK program, the character variable N$ can be assigned the value 'BIG OIL'. The character value assigned to a character variable must be enclosed in quotation marks:

```
115 N$ = 'BIG OIL'
```

In the PRINT statement, you indicate that N$ is to be included in the print-out:

```
130 PRINT  N$ '    STOCKPRICE=' P '    DIVIDEND=' D '    YIELD=' Y '    PERCENT'
```

When the PRINT statement is executed, the system prints BIG OIL, the value assigned to N$.

EXERCISE 4 - 2:

1. LIST the STOCK program.
2. Insert another LET statement in the STOCK program to assign a character value (company name) to the character variable N$. Use line number 115.

3. Modify the PRINT statement so that the value assigned to the character variable N$ will be printed.
4. LIST
5. RUN
6. Change the values in the program to the following:

   P = 12.40
   D = 1.00
   N$ = 'ANY UTILITY'
7. RUN the program with the new values.
   Your answer should be:

```
ANY UTILITY    STOCKPRICE=12.4    DIVIDEND=1    YIELD=8.06451  PERCENT
```

Note: Low order zeros are truncated with use of the PRINT statement. In the section "Formatting Results", a print statement that allows retention of zeros is discussed.

INPUT AND GO TO STATEMENTS

Modifying and rerunning the program each time new data is to be processed can be tedious and time-consuming. By replacing the LET statements with an INPUT statement and adding a GO TO statement, new values for the variables can be entered during execution of the program.

In the INPUT statement the variables in the program are listed. These variables must be separated by a comma:

```
0100 INPUT N$, P, D
```

This example contains a character variable (N$) and 2 arithmetic variables (P and D). When the INPUT statement is encountered during program execution, the system prints a question mark, indicating that values are needed for the variables. A value for each variable listed in the INPUT statement is then entered. These values must be separated by a comma.

```
'BIG OIL', 77.25, 4.20
```

Note: The character variable must be enclosed in quotation marks.

The values entered are assigned to the variables in the sequence the variables are listed in the INPUT statement, the first value entered being assigned to the first listed variable, the second value to the second variable, and so on.

After entering the data, press the RETURN key. Processing then continues using the newly entered variables.

The GO TO statement changes the normal execution sequence of a program. It tells the system to go to a particular line number instead of the next sequential line.

EXAMPLE: 130 GO TO 100

This statement instructs the system to go to statement 100 instead of the next sequential statement following line 130. In the STOCK program, the GO TO statement should indicate the line number of the INPUT statement so that values for the variables can again be entered. It should be placed before the END statement to prevent the program from terminating.

STOCK program

```
0100  INPUT N$, P, D
0110  Y = D * 100/P
0120 PRINT N$, 'STOCKPRICE=' P'DIVIDEND=' D'YIELD=' Y 'PERCENT'
0130  GO TO 100
0140  END
```

Terminating the Program

Because of the GO TO statement in this program, the system never reaches the END statement and the program never terminates. You must terminate the program.

The program can be terminated when the system prints a question mark to request data. First, on the Console lift the INQUIRY REQUEST switch, which puts the system in a pause status. Then key the GO command in the following format:

GO ABORT

The GO command entered in this format causes the system to terminate the program and return to the input status (READY). While the system is in the pause status (before GO ABORT is entered), statements in the Work File cannot be modified. Statements can be changed, added, or erased only when the system is in the READY status.

EXERCISE 4 - 3:

1. LIST the STOCK program.
2. Modify it by replacing the LET statements with an INPUT statement and adding a GO TO statement.
   a) The INPUT statement should be line 100.
   b) Lines 110 and 115 can be deleted by keying the line number and then pressing RETURN.
   c) The GO TO statement must precede the END statement.
3. RUN. Each time the system prints a question mark, enter a set of values:

N$	P	D
'BIG OIL'	25.75	1.10
'NEW DEVICE'	32.50	.75
'NIX POLLUTION'	15.60	.60

4. Terminate the program.

DATA AND READ STATEMENTS

A program that requests input data during execution is ideal for those problems in which all of the data is not available when the program is entered. For example, the STOCK program with the INPUT and GO TO statements can be run again without changing the program when the next annual dividends are known.

But suppose that your data is complete, and you wish to compute the yield based only on last year's dividends. Sitting at the keyboard and entering data each time the question mark appears is inconvenient. The job would be easier if you could enter all the values you wanted to process before execution began. This can be done with the DATA and READ statements.

By using the DATA statement, all of the values you want to process become part of the program in the Work File. As many DATA statements as needed to include all the data may be used. When one line becomes filled, continue the data on another line. Data items must be separated by a comma.

EXAMPLE:

```
100 DATA 'BIG OIL', 25.75, 1.10, 'NEW DEVICE', 32.50, .75
110 DATA 'NIX POLLUTION', 15.60, .60, 'ANY UTILITY'
120 DATA 12.40, 1.00, 'MINI AUTO', 29.25, .85
.....
.....
```

When the program is compiled, all data items from all DATA statements, regardless of where they are located in the program, are collected in one single data file.

The data items are arranged within the file in line number order. The system sets a data pointer to the first data element of the file, and an END OF FILE indicator after the last element of the file.

The READ statement instructs the system to read and assign this data to the variables specified in the statement.

```
130 READ N$, P, D
```

The system reads as many data items as required to assign a value to each variable in the READ statement. As values are assigned, the data pointer, originally set to the first element of the data file, moves ahead. In the STOCK program, since the READ statement has three variables, the system reads and assigns the first three values in the data file. The data pointer is then set at the next data item, that is, the fourth item of the file. After the values are assigned the system continues processing.

EXAMPLE:

```
120 DATA 'ANY UTILITY', 12.40, 1.00, 'MINI AUTO', 29.25, .80
130 READ N$, P, D
140 Y = D * 100/P
150 PRINT N$ 'STOCKPRICE=' P 'DIVIDEND=' D 'YIELD=' Y 'PERCENT'
160 GO TO 130
170 END
```

Line 160 of this program causes the READ statement to be executed again. The system reads three more values from the data file. Since the data pointer was set to the fourth item of the file in the STOCK program, items 4, 5, and 6 are read and assigned to the variables. These values are then processed. The system continues in this way until the last data element of the file has been assigned and the data pointer is set on END OF FILE.

Once the data pointer is set at END OF FILE, no additional values can be assigned. When the system returns to the READ statement and cannot assign any more values, an execution error occurs. The system prints the message:

ERROR 721 INSUFFICIENT DATA FOR READ

and returns to the input status. Since the system never reaches the END statement, this is how this STOCK program terminates. An execution error also occurs if the variable and the value assigned to it are not the same type of data. This happens if a character data item in the data file is assigned to an arithmetic variable in the READ statement, or vice versa. The system prints a message indicating this error has occurred:

ERROR 727 INVALID VARIABLE ASSIGNMENT

Since an invalid variable assignment terminates execution and returns the system to the input status, special care must be taken to maintain correspondence between values and the variables to which they are assigned. This is particularly true when there is a long list of variables in the READ statement or when more than one READ statement is used.

EXERCISE 4-4:

1. LIST the STOCK program.
2. Modify using DATA statements to enter the data and the READ statement to assign these values to the variables.

N$	P	D
'BIG OIL'	25.75	1.10
'NEW DEVICE'	32.50	.75
'NIX POLLUTION'	15.60	.60

3. RUN

Changing the Name of a Program

When the EDIT STOCK command was entered, the STOCK program was copied from the library into the Work File. Since then you have changed the program in the Work File so that it is no longer the same as the STOCK program in the library.

If you now enter SAVE, the program in the Work File is saved as the STOCK program. The old STOCK program in the library then becomes inaccessable.

If you save the program in the Work File under a different name, both programs would be available in the library. To do this, enter the SAVE command with the filename you want the program in the Work File to be associated with.

EXAMPLE: SAVE STOCK2

The STOCK program you originally saved will be available in the library under the name STOCK. The modified version can be referenced by the name STOCK2.

EXERCISE 4 - 5:

1. SAVE the program now in the Work File with the name STOCK2.
2. EDIT STOCK and LIST.
3. EDIT STOCK2 and LIST.

RESTORE STATEMENT

The data file created with the DATA statement can be used again in the program by moving the data pointer back to the beginning of the data file. The data pointer is reset with the RESTORE statement.

With the pointer back at the beginning of the file, the data items can again be assigned to variables specified in a READ statement.

EXAMPLE:

```
100 DATA 10, 20, 30
110 READ A, B, C
120 X = A * B / C
130 PRINT X
140 RESTORE
150 READ D, E, F
160 Y = D * E * F
170 PRINT Y
180 END
```

When the READ statement in line 110 is executed, the values 10, 20, and 30 are assigned to A, B, and C respectively. Execution of the RESTORE statement in line 140 resets the data pointer so that the values 10, 20 and 30 are reused to assign values to the variables D, E, and F.

EXERCISE 4 - 6:

1. EDIT TEST
2. Enter the program shown above
3. RUN

The answers are:

6.66667
6000

Section 5 Assigning Values

Performance Objectives:

Assign values to variables with LET statements

Write BASIC expressions

Describe the BASIC arithmetic operators and the priority of arithmetic operations

THE LET STATEMENT

The LET statement assigns values to variables. In a LET statement, at least one variable name must be specified to the left of the = sign, and either an arithmetic expression or a character expression must appear to the right of the = sign. The keyword LET is optional.

Execution of the LET statement causes the computed value of the expression to the right of the = sign (represented internally in the system as a single quantity) to be assigned to each of the variables named to the left of the = sign. In BASIC the = sign means *assign*. It does not mean "equal".

If the = sign were an equal sign, then the equation A = A + 3 would be invalid. However, in BASIC this is a valid assignment statement. The value of the arithmetic expression A + 3 is computed using the presently assigned value for A. The newly computed value, which is 3 greater than the old value, is then assigned to A. In the exercise below, this type of assignment statement is used to calculate interest.

EXERCISE 5 - 1:

1. EDIT INT
2. Enter and execute the following program to calculate 3% interest on $1.00, compounded annually, for T years.

```
0100 C = 1
0110 T = 1
0120 C = C * 1.03
0130 PRINT 'AFTER' T 'YEARS, CAPITAL=' C
0140 T = T + 1
0150 GO TO 120
0160 END
RUN
```

Note: Lines 100 and 110 could be combined into one LET statement C, T = 1. When only one value is entered, the value is assigned to all variables in the statement.

3. Terminate execution by using the INQUIRY REQUEST switch and keying the GO ABORT command.

Either a character expression or an arithmetic expression may appear to the right of the = sign. A character expression may be:

a character constant	100 LET A$ = 'TOTAL'
a character variable	100 LET A$ = N$

When a character expression is used, the variable named to the left of the = sign must be a character variable. When an arithmetic expression is used, the variable named to the left of the = sign must be an arithmetic variable. The arithmetic expression to the right of the = sign may be:

an arithmetic constant	100 LET A = 100
an arithmetic variable	100 LET A = B
a series of arithmetic operations	100 LET A = A * B/3

Arithmetic Operators

The following BASIC symbols are used to indicate arithmetic functions.

OPERATOR	OPERATION
↑ or **	Exponentiation
*	Multiplication
/	Division
+	Addition
–	Subtraction

The symbols + and – can also be used to indicate positive or negative value.

EXAMPLE: 100 LET A = – 26

Shown below are examples of arithmetic expressions written in BASIC.

ARITHMETIC EXPRESSION	BASIC EXPRESSION
-45^3	–45 ↑ 3
$(a+b)^2$	(A + B) ↑ 2
$2ax - x^2$	2 * (A * X) – X ↑ 2
$a^2 - \frac{a^2 y^2}{b^2}$	A ↑ 2 – (A ↑ 2 * Y ↑ 2)/B ↑ 2
$a^{(1/2)}$	A ↑ (1/2)
$\frac{ab}{a+b}$	A * B / (A + B)
$\left(\frac{ab}{a+b}\right)^2$	((A * B) / (A + B)) ↑ 2
$\sqrt[3]{(a+b)}$	(A + B) ↑ (1/3)
$ax^n + a_1x^{n-1} + a_2x^{n-2} + a_3x^{n-3}$	A*X↑N+A1*X↑(N– 1)+A2*X↑(N–2)+A3*X↑(N–3

Levels of Priority

Operations in an arithmetic expression are executed by the system according to a hierarchy of operations. The priority of operations is given in the following table:

LEVEL 1	Operations within parentheses
LEVEL 2	** or ↑ Exponentiation
LEVEL 3	*, / Multiplication and Division
LEVEL 4	+, – Addition and Subtraction

When operations in an expression have equal priority, for example addition and subtraction, the operations are performed from left to right. The following program illustrates equal priority level:

```
0100 C = –4
0110 B = 2
0120 X = 3+B - C+5
0130 PRINT X
0140 END
```

Computation of the expression 3+B – C+5

Step 1:	3 + B = 5	3 + 2 = 5
Step 2:	5 – C = 9	5 – (-4) = 9
Step 3:	9 + 5 = 14	

PRINTED RESULT:

14

In the program shown below, however, the expression is not executed from left to right. In BASIC, multiplication has a higher priority than addition, so multiplication is performed before addition.

```
0100 A = 10
0110 B = 5
0120 Y = 50 + A * B
0130 PRINT Y
0140 END
```

Computation of the expression 50+A*B

Step 1: A * B = 50
Step 2: 50 + 50 = 100

PRINTED RESULT:

100

Operations within parentheses are always performed first. Parentheses indicate a subexpression and are an instruction to perform the operation indicated in the subexpression before executing the rest of the expression.

EXAMPLE: 100 Z = (A*B)↑ 2/C

In any one expression, up to 8 nested parentheses levels may be used. When subexpressions are contained within subexpressions, the deepest nested expression is computed first.

EXAMPLE:

```
130 N = ((((((((A9*B+A8)*B+A7)*B+A6)*B+A5)*B+A4)*B+A3)*B+A2)*B+A1)*B+A
```

The following exercise shows the effect of parentheses within a given expression.

EXERCISE 5 - 2:

1. Clear the Work File and name it EXP.
2. Enter the following program:

```
0100 A = 6
0110 B = 5
0120 C = 4
0130 R = A * B + C
0140 PRINT 'ANSWER IS' R
0150 END
```

3. RUN
4. Change statement 130 to
   0130 R = A * (B + C)
5. Execute the program. Compare the results.

Numeric Data

To represent numbers, 14 different characters are used:

0 through 9
the decimal point '.'
the signs + and –
the letter E

Whole numbers, decimal numbers, and numbers in E-notation may be entered. These numbers may be preceded by a sign. The E-notation is especially useful when entering very large or very small values.

Number	Number in E-notation
100000000	10E+7
.00000123456	1.23456E-6
.000037	3.7E-5 or 37E - 6 or 370E-7
-560.45	-5.6045E2
-.0000560	-5.6E-5

Any number of digits may be entered to represent a value but all values must be within the range of 10^{99} and 10^{-99}. An attempt to enter a number beyond these limits results in a syntax error.

Internally, numbers are represented in E-notation and calculated in either SHORT precision (7 significant digits) or LONG precision (15 significant digits). The precision used is the precision specified with an execution command. For example, the RUN command can be keyed in either of the following ways to indicate the precision.

```
RUN-LONG
RUN-SHORT
```

If the precision is not specified, the system assumes short precision.

Precision in Arithmetic Expressions

The precision defined for the program (SHORT or LONG) is used to compute the value of an arithmetic expression. When necessary, the value is rounded to either 7 or 15 significant digits after each operation within the expression. Whenever the utmost defined precision is necessary, consideration should be given to the form in which an arithmetic expression is written. Due to rounding, the expression A*B*C does not necessarily produce the same result as A* (B*C).

System Constants

Three frequently used arithmetic constants are provided with the system. These are: pi (ratio of the circumference of a circle to its diameter), e (base of a natural logarithm), and $\sqrt{2}$ (positive square root of 2).

The values of these constants and the names by which they are referenced are shown below:

Name	SHORT Precision	LONG Precision
&PI	3.141593	3.14159265358979
&E	2.718282	2.71828182845905
&SQR2	1.414214	1.41421346237310

These system constants can be used anywhere that a normal arithmetic constant can be used, and can be preceded by a plus or minus sign. The system provides the value of the system constant when it encounters the name in a program.

Section 6 Branching

Performance Objectives:

Control the sequence of program execution with GO TO. . .ON statements

Control the sequence of program execution with IF. . .THEN statements

GO TO . . . ON STATEMENT

A branch is an instruction that changes the normal line-number execution sequence of a program. The GO TO statement used in the STOCK program created an *unconditional branch.* It transferred execution to the segment of the program beginning with the line number specified in the GO TO statement.

EXAMPLE: 200 GO TO 150

Only one line number can be specified in the GO TO statement. With the GO TO. . . ON statement, however, more than one line number may be listed so that the program can be branched to one of several locations in the program. The GO TO . . . ON statement creates a *conditional branch.* It allows you to describe a condition that will control the branch.

EXAMPLE: 10 GO TO 50, 100, 150 ON R

The branch is controlled by the value of the arithmetic expression following ON. When the value of the expression is 1, the program branches to the first line number listed. When the value is 2, the branch is to the second line number. A value of 3 branches the program to the third listed line number, and so on. If the value is a decimal fraction, then only the integer part of the value is used. A value whose integer part is negative or zero or greater than the number of line numbers listed is ignored and execution continues with the next statement in line number sequence.

Shown below is the structure of a program in which the GO TO. . . ON statement is used to branch to various segments of the program.

EXAMPLE:

```
100 Root, or main, segment of the program
 :  :  :
250 INPUT A
260 GO TO  290, 400, 510, 620, 100 ON A
270 PRINT 'ENTERED VALUE OUTSIDE OF RANGE'
280 GO TO 250
290 Segment 1 of program
 :  :  :
390 GO TO 100
400 Segment 2 of program
 :  :  :
500 GO TO 100
510 Segment 3 of program
 :  :  :
610 GO TO 100
620 Segment 4 of program
 :  :  :
720 GO TO 100
```

After the root segment is executed, the INPUT statement is encountered and a question mark is printed, indicating that a value for A should be entered. When the value entered for A is 1, 2, 3, or 4, the program branches to 290, 400, 510, and 620 respectively. At the end of each of these program segments, the program continues with line 100.

When the value of A is 5, the program returns immediately to line 100. When the value is less than one, or greater than 5, the program continues in line-number sequence, printing ENTERED VALUE OUTSIDE OF RANGE and then returning to line 250.

In the "Computational Test" program that follows, a branch occurs if the expression following ON has a value ranging from 1 through 3. If the value of the expression is greater than 3 or less than 1, the program automatically continues in line-number sequence.

```
100 PRINT 'A COMPUTATIONAL TEST'
110 PRINT 'WHEN THE ? IS PRINTED, ENTER A VALUE.'
120 PRINT 'WHAT IS THE SUM OF 3 + 4'
130 INPUT R
140 GO TO 170, 190, 210 ON R-5
150 PRINT 'YOU MUST HAVE FORGOTTEN THE QUESTION. READ IT AGAIN.'
160 GO TO 120
170 PRINT 'YOU''RE ALMOST RIGHT BUT STILL A LITTLE LOW. TRY AGAIN'
180 GO TO 130
190 PRINT 'BINGO'
200 GO TO 230
210 PRINT 'NOT BAD BUT A LITTLE HIGH.  TRY AGAIN'
220 GO TO 130
230 END
```

If the number 6 is entered as the value for R, the value of the expression following ON (R - 5) is 1, and a branch to line 170, the first line number listed, occurs. If the value for R is 7, then R - 5 is 2 and the program is branched to the second line number listed (line 190). The expression R - 5 is 3 when the value of R is 8. In this case the branch is to line 210, the third listed line number. Execution continues in line number sequence, that is, with line 150, if the value of R - 5 is less than 1 or greater than 3.

EXERCISE 6 - 1:

1. Enter the "Computational Test" program shown above.
2. RUN

THE IF . . . THEN STATEMENT

The IF . . . THEN statement is also used for conditional branching.

EXAMPLE: IF A <= B THEN 240

The first part of the statement contains the IF - expression. It consists of two expressions separated by a relational operator. The relational operator describes the condition that is to be tested.

Following are the BASIC relational operators:

Relational Operator	Condition
<=	less than or equal to
>=	greater than or equal to
≠	not equal to
<>	not equal to
<	less than
>	greater than
=	equal to

When the system executes the IF. . .THEN statement, it first compares the two expressions. If the relational operator describes the actual relationship between the two expressions, then the IF - expression is true, and the program is branched to the line number following the keyword THEN. If the relational operator does not describe the actual relationship between the two expressions, then the IF - expression is false and execution continues with the next statement in line number sequence.

The two expressions compared in the IF - expression must be the same type of data. That is, they both must be either arithmetic or character.

Arithmetic Expressions in IF - expressions

For arithmetic expression to be true, both expressions must fulfill the described condition over the entire length of their specified precision, regardless of their representation or entered format. For example, 1 is equal to 1 and also to 1.000000. But 1 or not equal to 1.000001.

The following are examples of true and false arithmetic IF - expressions.

EXAMPLE:

IF - expressions	Values	True or False
A = B	A is .431568 B is 43.1568E-2	True
A - B<= 0	A is 5 B is 6	True because A - B = -1
A + B ≠ 0	A is 5 B is -5	False because A + B = 0

Character Expressions in IF - expressions

Before character expressions are compared, character constants containing less than 18 characters are expanded on the right with blanks to an overall length of 18. Character constants containing more than 18 characters are truncated to 18. The comparison takes place character by character from left to right. The true or false decision is made at the leftmost position where the comparison is "not-equal".

In the following example, the IF - expression is true when the character variable N$ has the exact value 'END OF FILE'.

```
IF N$ = 'END OF FILE' THEN 100
```

But if the value of N$ contained an extra blank between END and OF, the IF - expression would be false.

```
IF N$ = 'END OF FILE' and N$ is 'END    OF FILE'
```

In this case, the IF - expression becomes false when the comparison is made at the 5th position of the character string.

The IF . . .THEN statement can be used in the STOCK2 program to print out only those companies whose capital gain (Y) is greater than or equal to 4.5%. (The STOCK2 program has the DATA and READ statements.)

```
0100 DATA 'BIG OIL', 25.75, 1.10, 'NEW DEVICE', 32.50, .75
0110 DATA 'NIX POLLUTION', 15.60, .60, 'ANY UTILITY',
0120 DATA 12.40, 1.00, 'MINI AUTO', 29.25, .85
0130 READ N$, P, D,
0140 Y = D * 100/P
0145 IF Y < 4.5 THEN 130
0150 PRINT N$ 'STOCKPRICE=' P 'DIVIDEND=' D 'YIELD=' Y 'PERCENT'
0160 GO TO 130
0170 END
```

After the value of Y is calculated, the value is compared to 4.5. If the computed value of Y is equal to or greater than 4.5, the values specified in the PRINT statement are printed before the program is branched to line 130. If the value of Y is less than 4.5, then the program is immediately branched to line 130.

EXERCISE 6 - 2:

1. EDIT STOCK2
2. LIST
3. Add the IF . . .THEN statement to the program so that only those companies whose capital gain is greater than or equal to 4.5% will be printed.
4. LIST
5. RUN
6. Do not enter the SAVE Command and do not clear the Work File.

Practice in Programming

After the results of a program have been produced, it is sometimes desirable to perform calculations on these results. For instance, in the STOCK2 program a total of all the stock prices and all the dividends could be obtained to calculate an average yield based on these totals.

These calculations require another program segment to which the system can branch when all the values for P, D, and Y have been processed. In the STOCK2 program, however, this presents a problem. As soon as data can no longer be read from the data file, the error message

ERROR 721 INSUFFICIENT DATA FOR READ

is printed and the program is terminated. The program can be kept from terminating at this point by adding some unique data at the end of the data file and using the IF ... THEN statement. Data groups in the STOCK program data file consist of a character data item and two numeric values. The following data group can be entered as the last data group in the file:

'END OF DATA', 0, 0

An IF. . .THEN statement to check whether or not the last data item has been read is also added to the program:

IF N$ = 'END OF DATA' THEN 500

If the data just read equals 'END OF DATA', the program branches to 500, the specified line number. This line number will indicate the statement to calculate average yield:

500 Y9 = D9 * 100/P9

The variables Y9, D9, and P9 are arbitrary names for the average yield, the total dividend, and the total stock price. This LET statement will be followed by a PRINT statement causing the totals and the average yield to be printed. An END statement after this PRINT statement terminates the program.

Two more LET statements are needed to accumulate stockprice and dividend totals. These two LET statements should be included after the READ statement but before the GO TO statement that unconditionally branches back to the READ statement.

EXAMPLE:

D9 = D9 + D
P9 = P9 + P

EXERCISE 6 - 3:

1. Modify the STOCK program now in the Work File so that total stockprice, total dividends, and the average yield will be printed.
2. LIST
3. RUN
4. Do not SAVE and do not clear the Work File.

STOCK program with segment for computing average yield:

```
0100 DATA 'BIG OIL', 25.75, 1.10, 'NEW DEVICE', 32.50, .75
0110 DATA 'NIX POLLUTION', 15.60, .60, 'ANY UTILITY',
0120 DATA 12.40, 1.00, 'MINI AUTO', 29.25, .85
0125 DATA 'END OF FILE', 0, 0
0130 READ N$, P, D
0135 IF N$ = 'END OF FILE' THEN 500
0140 Y = D * 100/P
0145  P9 = P9 + P
0148  D9 = D9 + D
0150 PRINT N$ 'STOCKPRICE=' P 'DIVIDEND=' D 'YIELD=' Y 'PERCENT'
0160 GO TO 130
0500 Y9 = D9 * 100/P9
0510 PRINT 'TOTAL STOCKPRICE IS' P9 'DIVIDENDS' P9 'AVERAGE YIELD' Y9
0520 END
```

WRITING COMMENTS

Ordinarily, only two types of lines are entered: statement lines and command lines. However, it is often desirable to include comments within a program to provide clarification or explanation. Comments can be included in the following ways:

1. A comment line is entered by starting the line with an asterisk. These lines are not stored and do not become an integral part of the program.

```
*THIS IS A LINE STARTING WITH AN ASTERISK.
*IT DOES NOT HAVE ANY EFFECT ON THE SYSTEM.
```

2. The REM (Remark) statement is also used to include desired comments. The REM statements begins with a line number, and the text that follows the keyword REM becomes part of the program. The statement itself is non-executable, and the text is not printed during program execution. The text is printed only when the program is listed.

```
100 DATA 'MINI AUTO', 29.25, .85 'END OF FILE', 0, 0
110 REM LINE 140 HAS A CONDITIONAL
120 REM BRANCH SO THAT TOTALS AND
130 REM AN AVERAGE YIELD WILL BE COMPUTED.
140 IF N$ = 'END OF FILE' THEN 500
```

Lines 110 through 130 are printed only when the LIST command is entered.

3. A PRINT statement is used to print a comment during execution.

```
130 PRINT 'TOTAL CALCULATIONS APPEAR BELOW'
```

EXERCISE 6 - 4:

1. Modify the program now in the Work File to include a remark explaining the conditional branch.
2. SAVE the program. (In your library, this modified program becomes associated with the name STOCK2. The old STOCK2 program is no longer accessable.)

Section 7 Program Loops

Performance Objectives:

- Create repetitive program operations
- Write programs using nested loops

PROGRAM LOOPS

The repeated execution of a segment of a program allows the same calculations to be performed on different groups of data. Consider, for example, the following program to compute compound interest. In it lines 110 - 140 are executed repeatedly, compounding interest for an indefinite number of years:

```
0100 C, T = 1
0110 PRINT 'AFTER' T 'YEARS, CAPITAL IS' C
0120 C = C * 1.03
0130 T = T + 1
0140 GO TO 110
0150 END
```

To terminate this program, it is necessary to lift the INQUIRY REQUEST switch, and key the GO ABORT command. No control is provided for terminating the program when the desired amount of data has been processed, for example, when interest has been compounded for 35 years.

If the program is to terminate after interest has been computed for 35 years, the IF. . .THEN statement can be used to control the variable T:

```
0140  IF T < = 35 THEN 110
```

This statement causes the program to branch back to line 110 if T is less than or equal to 35. The variable T controls the repeated execution of the program segment. Controlled repetition of the same program segment is referred to as a *program loop.*

BASIC provides a pair of statements especially for controlling a program loop. These are the FOR and NEXT statements, shown in the program below:

```
                0100  C = 1
              ┌►0110  FOR T = 1 TO 35 STEP 1
Program       │ 0120  C = C * 1.03
Loop          │ 0130  PRINT 'AFTER' T 'YEARS CAPITAL IS' C
              └►0140  NEXT T
                0150 END
```

The FOR statement in line 110 begins the program loop, and the NEXT statement in line 140 ends the loop. The program segment will be executed 35 times, as indicated in the FOR statement where the *control variable* T is defined with an *initial value* of 1 and a *final value* of 35. The value following the keyword STEP in the statement is the *increment.* Each time the segment is executed the value of the control variable T is increased by 1.

The first time a loop is entered, the control variable is set to the initial value, that is, the value of the expression following the = sign in the FOR statement. Then the statements in the loop are executed. When the NEXT statement is encountered, the value of the expression following the keyword STEP is calculated and added to the control variable. The control variable is then compared to the final value.

In the compound interest example, the loop is again executed only if the control variable is smaller than or equal to the final value. If the control variable is greater than the final value, the program leaves the loop and execution continues with the line number following the NEXT statement.

Other programs, however, might require the control variable to be equal to or greater than the final value before the loop is executed:

```
0300  FOR X = 10 TO 1 STEP -1
. . .
. . .
0400  NEXT X
. . .
. . .
```

Here the initial value is greater than the final value, and the increment is negative. The loop is executed only if the control variable is greater than or equal to the final value. With each execution of the loop, the control variable is reduced because of the negative increment. When the value of the control variable becomes smaller than the final value, execution continues with the line number following the NEXT statement. Whenever the initial value is greater than the final value, the increment must be negative or the loop will never be executed.

Any valid arithmetic expression may be used as the initial value, the final value, and the increment.

```
0100 FOR P4 = (A - B) ↑2/4 TO X4 STEP B↑2
. . .
. . .
. . .
0250 NEXT P4
```

In the FOR statement, the keyword STEP followed by an arithmetic expression may be omitted. If no increment is provided, a value of 1 is assumed. In the compound interest program, the FOR statement could have been written as:

```
0110  FOR T = 1 TO 35
```

Practice in Programming

The FOR and NEXT statements can be used to determine the mean average test score for each of the following students:

```
Bill   98, 95, 100
Tom    82, 96, 91
Jim    72, 65, 98
Joe    75, 61, 75
```

The following variable names are used in the program:

```
N$   student name
C    control variable
R    student test score
T    total of all student scores
A    average for each student
```

Data is entered with the DATA statement:

```
100 DATA 'BILL', 98, 95, 100, 'TOM', 82, 96, 91
```

Two READ statements are required. The first is used to read a student name:

```
120 READ N$
```

The second READ statement is within a loop and causes one test score to be read each time the loop is executed:

```
150 READ R
```

A LET statement is also within the loop to total the scores for each student:

```
160  T = T + R
```

Another LET statement, outside the loop, is used to calculate the average:

```
180 A = T/C
```

The complete program appears below:

```
0100  DATA 'BILL', 98, 95, 100, 'TOM', 82, 96, 91, 'JIM'
0110  DATA 72, 65, 98, 'JOE', 75, 61, 75
0120 READ N$
0130 T = 0
0140  FOR C = 1 TO 3
0150  READ R
0160  T  =  T + R
0170  NEXT C
0180  A  =  T/C
0190  PRINT 'AVERAGE FOR' N$  'IS' A
0200  GO TO 120
0210  END
```

EXERCISE 7 - 1:

1. EDIT the Work File using a filename of AVERAGE.
2. ENTER the program.
3. RUN
4. SAVE

More Practice in Programming

In the previous example, AVERAGE, each student had 3 test scores. This made it possible to assign a value of 3 to C, the control variable. However, if this example is changed to allow each student a variable number of test scores, then C cannot be assigned a fixed value. The number that is assigned to C must be the number of test scores given for each student. For example, if Bill has 4 scores and Tom has only 3, then the value of C must be 4 for Bill and 3 for Tom. If Q is assigned to represent the variable number of scores, the control statement is modified as follows:

```
140 FOR C = 1 TO Q
```

Before the loop can be executed, the value of Q must be provided. This value is to be included in the data file created by the DATA statement and must be read before the first execution of the FOR statement. It can be read at the same time the student name is read:

```
120  READ N$, Q
```

In the DATA statement, then, a value for Q is entered following every character constant (student name). For Bill this value is 4, for Tom 3.

EXAMPLE:

```
0100  DATA 'BILL', 4, 98, 95, 100, 80, 'TOM', 3, 82, 96, 91
0110  DATA 'JIM', 5, 72, 65, 98, 90, 82, 'JOE', 4, 75, 61, 75, 78
```

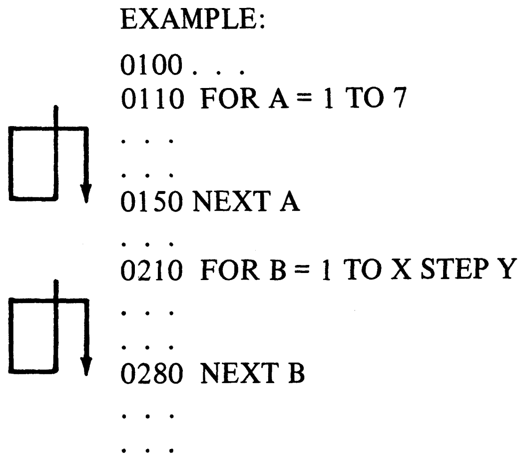
EXERCISE 7 - 2:

1. EDIT the previously saved AVERAGE program.
2. LIST the Work File.
3. Modify this program by replacing the DATA statements with those shown in the example above and modify statements 120 and 140 to allow a variable number of test scores.
4. RUN
5. SAVE this program for use in a later exercise. Use the name AVG in the SAVE command to prevent destroying the previously saved AVERAGE program.

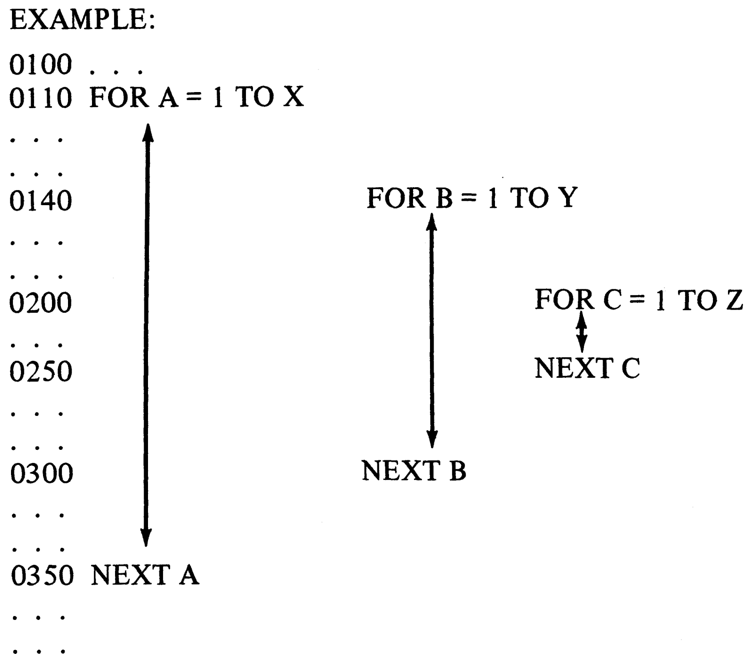
LOOPS IN SERIES AND NESTED LOOPS

A program may contain more than one program loop. These loops can be either nested or in a series.

Loops are *in a series* when one program loop is entered only after the preceding loop has been concluded normally, that is, concluded as a result of the comparison between the control variable and the final value.



Loops are *nested* when one or more complete loops are contained within another loop.



Here the FOR B. . . NEXT B loop is nested in the FOR A. . . NEXT A loop; and the FOR C. . . NEXT C loop is nested in both the FOR B. . . NEXT B loop and the FOR A. . . NEXT A loop. In nested loops, another loop is entered before the preceding loop is concluded.

FOR loops may be nested to a depth of 9 loops, counting the outer loop. If this limit is exceeded, a compilation error occurs, and an error message is printed.

Using nested loops, a very short program can produce a multiplication table (multiplicand x multiplier = product) for any specified number of factors. In the program below, the multiplicand (B) has a range from 1 through 10 and the multiplier (A) from 1 through 12.

```
0100 FOR B =1 TO 10
0110 FOR A = 1 TO 12
0120 PRINT B 'X' A '=' B * A
0130 NEXT A
0140 PRINT
0150 NEXT B
0160 END
```

The program has two loops, one nested in the other. The FOR B . . . NEXT B loop increases the value of the multiplicand (B) to a maximum of 10. The FOR A . . . NEXT A loop increases the multiplier (A) until a maximum value of 12 is reached. With each execution of the FOR B . . . NEXT B loop, the nested FOR A . . . NEXT A loop is executed 12 times.

Two facilities of the PRINT statement not yet discussed are employed in the program. In a PRINT statement, any valid expression may be used in place of a variable. The PRINT statement in the FOR A . . . NEXT A loop uses the expression B * A instead of a variable name, for instance C, for the product. Using the arithmetic expression (B * A) eliminates the need for a LET statement, C = B * A, which would have to be inserted before the PRINT statement.

A PRINT statement in which only the keyword PRINT is used causes the printer to space one line. Because of the PRINT statement in line 140, the printed result will have blank lines dividing the multiplication table whenever the value for the multiplicand (B) changes.

EXERCISE 7 - 3:

1. Enter and execute the program for creating a multiplication table.
2. Modify and then execute the program so that a division table will be printed.

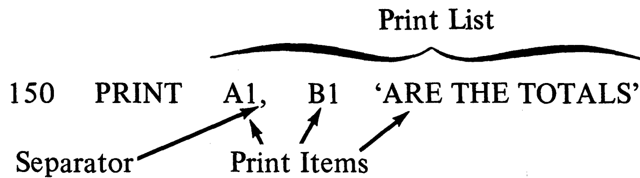
Section 8 Printing Results

Performance Objectives:

- Control the width of print lines
- Control vertical line spacing
- Control horizontal spacing

THE PRINT STATEMENT

Execution of a PRINT statement causes data to be printed. Printing takes place from left to right, controlled by the *print list* following the keyword PRINT in the statement. Arithmetic expressions, character expressions, and character constants in the print list are referred to as *print items.* Commas and semicolons are used as *separators* in the print list. They are not printed but used for horizontal and vertical spacing.



Defining the Width of a Print Line

The amount of information that can be contained in one line is determined by the number of print positions available in the print line. Depending on the model, the printer on the IBM System/3 Model 6 has either 132 or 220 print positions. All print positions or only a specified number of these can be made available for printing.

The WIDTH command may be entered to specify the number of print positions to be used. The specified width cannot be less than 18 or greater than the number available with the printer (132 or 220). The size of the left margin in print positions can also be specified.

EXAMPLE:

```
WIDTH 122
WIDTH 90
WIDTH 122, 10
```

In the first example, the print line is defined as 122 print positions beginning at print position 1. In the third example, the WIDTH command defines a 122 print position line that begins at print position 11.

The margin parameter of the command is optional. If omitted, the present left margin is not affected.

EXERCISE 8 - 1:

1. For a printer with 132 print positions, enter the following WIDTH command:

   WIDTH 122, 10

PRINT Statement Without a Print List

As mentioned earlier, a PRINT statement without a print list causes the printer to space one line. The printhead returns to the left margin if it is at any other position when spacing occurs. The three PRINT statements shown below will cause the print paper to move up 3 lines.

```
EXAMPLE:
. . .
. . .
0200 PRINT
0210 PRINT
0220 PRINT
. . .
```

The Print List

A print list is printed from left to right, starting at the left margin of the printer. For each print item, the data associated with that item is printed. A comma or a semicolon in the print list causes spacing between the data items.

```
EXAMPLE:
0800 PRINT 900/30, 100/25

PRINTED RESULT:
30                4
```

Commas as Separators

For each comma in the print list, the printhead spaces to the right. If the comma preceded by a print item, the print head spaces until it is 18 positions past the first character of the print item. Study the following PRINT statement:

```
0200 PRINT A, B ' ARE THE TOTALS'
```

Suppose the value of A is 50 and the value of B is 35. The value of A (50) will be printed beginning at the left margin, in position 1. Then because of the comma, the printhead will space to print position 19, that is, 18 positions beyond the first character of the value of A to print the value of B.

The value of B will be printed beginning in position 19. Since there is no comma after B, the character constant 'ARE THE TOTALS' will begin in the print position following the printed value for B. But because the first character of this character constant is a blank, the printed result will show one space between the value of B and the character constant.

```
50                35 ARE THE TOTALS
```

The 18-print position block created by the comma is referred to as a *long print zone.* This function is especially useful when the printed result is to be a table with columns.

EXAMPLE:

```
. . .
0210  B = 2
0220  FOR A = 1 TO 50
0230  PRINT A, B, A * B
0240  NEXT A
. . .
```

Because of the commas in the PRINT statement, the values for B will always be printed 18 positions past the first character of the values for A. Whether A is a 1-digit value or a 2-digit value, the position for the value of B never changes.

And the values for A * B will always begin 18 positions beyond the first character of the values for B. Using commas assures that the values in each column will always begin in the same print position.

PRINT OUT:

```
1                 2                 2
. . .             . . .             . . .
9                 2                 18
10                2                 20
11                2                 22
. . .             . . .             . . .
50                2                 100
```

In the next example, the first item in the print list is a comma. When the statement is executed, the printhead moves 18 spaces to the right of the left margin before printing the character constant 'TOTAL'.

EXAMPLE:

```
0150 PRINT   , 'TOTAL'
```

PRINTED RESULT:

```
                  TOTAL
```

Commas in a print list may also be used in a series . In the following example, the first comma causes the printhead to move 18 positions beyond the first character of the value A. The second comma moves the printhead 18 more spaces to the right before the value B is printed, beginning in position 37.

EXAMPLE:

```
. . .
0410  A, B = 2650
0420  PRINT A, , B
```

PRINTED RESULT:

```
2650                                2650
```

Practice in Programming

Write a program that will produce the following information for the values 1 to 35: the value (N), the reciprocal value (1/N), the square of the value (N↑2), the circumference of a circle with the value as the diameter (N*&PI), and the area of a circle with the value as the diameter (N/2↑2*&PI). The printed result should be a 5-column table with these headings:

N 1/N N↑2 N*&PI (N/2) ↑2*&PI

To create the columns, five long print zones should be used. One blank line should separate the headings from the rest of the column.

In the PRINT statement that creates the headings, N, 1/N, N↑2, N*&PI, and (N/2) ↑2*&PI are used as character constants. This PRINT statement should be the first statement in the program and should be followed by another PRINT statement that causes the printer to space one line.

In a third PRINT statement, N, 1/N, N↑2, N*&PI, and (N/2) ↑2*&PI are used as arithmetic expressions. Using arithmetic expressions eliminates the need for LET statements. This last PRINT statement should be within a FOR . . . NEXT loop. Write the program and then compare it with the following program.

EXERCISE 8 - 2:

1. Enter this program with the name TABLE:

```
0100 PRINT 'N', '1/N', 'N↑2', 'N*&PI', '(N/2)↑2 *&PI'
0110 PRINT
0120 FOR N = 1 TO 35
0130 PRINT N, 1/N, N↑2, N*&PI, (N/2)↑2*&PI
0140 NEXT N
0150 END
```

2. Execute the program in short precision. The command to accomplish this is: RUN-SHORT or RUN
3. Execute the TABLE program in long precision (RUN-LONG)
4. SAVE

Controlling the Printhead Position

After execution of a PRINT statement, the printhead can return to the left margin of the following line or remain in its present position in the same line. The position of the printhead following execution of a PRINT statement is determined by the last item in the print list.

If the last item is a print item (arithmetic expression, character expression, or character constant), the item is printed and the printhead moves to the leftmost position of the next line. If the last item is a comma, the printhead moves until it is 18 positions beyond the first character of the preceding print item and remains in that position. The print-out will then be continued on the same print line when the next PRINT statement is executed.

```
EXAMPLE:
. . .
. . .
0200 PRINT 'A',
. . .
0250 PRINT 'B',
. . .
0300 PRINT 'A * B'
PRINTED RESULT:
A                 B                 A * B
```

In this example, the output indicated by 3 different PRINT statements is printed in the same line. Each print item is contained within a long print zone. Of course, the same print-out could have been achieved with one PRINT statement:

```
0300 PRINT 'A', 'B', 'A * B'
```

The advantage of using more than one PRINT statement for print-outs in the same line is shown in the following practice problem.

Practice in Programming

Shown below is the AVG program for computing averages given variable numbers of scores.

```
0100 DATA 'BILL', 4, 98, 95, 100, 80 'TOM', 3, 82, 96, 91
0110 DATA 'JIM', 5, 72, 65, 98, 90, 82  'JOE', 4, 75, 61, 75, 78
0120 READ  N$, Q
0130 T = 0
0140 FOR C = 1 TO Q
0150 READ R
0160 T = T + R
0170 NEXT C
0180 A = T/C
0190  PRINT 'AVERAGE FOR' N$ 'IS' A
0200 GO TO 120
0210 END
```

Modify the program so that every score as well as the student name and average score will be printed. Each student name, his scores, and his average score should be printed on one line. Modify this program and then compare it with the program on the following page.

The PRINT statement that causes each name (N$) to be printed must be encountered before the FOR . . . NEXT loop is entered so that the student name prints only once. To prevent line spacing after the name is printed, the last item in the print list must be a comma. The PRINT statement that causes each score (R) to be printed is within the FOR . . . NEXT loop. The last item in this print list is also a comma. Finally, the PRINT statement in line 190 must be changed so that the student name will not be printed a second time.

Modified AVG Program:

```
0100 DATA 'BILL', 4, 98,95, 100, 80, 'TOM', 3, 82, 96, 91
0110 DATA 'JIM', 5, 72, 65, 98, 90, 82, 'JOE', 4, 75, 61, 75, 78
0120 READ N$, Q
0125 PRINT N$,
0130 T = O
0140 FOR C = 1 TO Q
0150 READ R
0155 PRINT R,
0160 T = T + R
0170 NEXT C
0180 A = T/C
0190 PRINT 'AVERAGE=' A
0200 GO TO 120
0210 END
```

EXERCISE 8 - 3:

1. EDIT the saved AVG program
2. LIST the program
3. Insert or modify the appropriate lines.
4. RUN

Continuing the Print-Out to the Next Line

Whenever the print-out cannot be contained in one line, the printer automatically spaces to the next line to continue the print-out.

Before each item in the print list is printed, a check is made to determine the number of spaces remaining in the line. When items in the print list are separated by commas, there must be at least 18 spaces remaining before a character item is printed. If fewer than 18 spaces remain, the character item is printed in the following line. For an arithmetic item, there must be enough spaces remaining to contain the item. Otherwise it is printed on the next line.

Semicolons as Separators

To print results as closely together as possible and still maintain readability and separation between print items, the semicolon can be used as a separator.

A semicolon between items in a print list creates a *short print zone.* This does not have a fixed length. Its length depends on the number of characters in the print item that precedes the semicolon.

When the preceding item is a character item, the short print zone is as long as the printed item. In other words, no spacing occurs after the item is printed.

When the preceding item is an arithmetic item, the length of the short print zone is the number of characters in the item plus a variable number of blank spaces. If, for example, the preceding arithmetic print item has 4 characters, the printhead moves 2 spaces to the right of the printed item to form a short print zone that is 6 positions in length. But if the preceding print item has 5 characters, the printhead moves 4 spaces to the right of the printed item to create a short print zone that is 9 positions in length.

Therefore, when the semicolon is used as a separator, it is difficult to arrange print items in columns unless all data items have the same length. A table describing the various sizes of short print zones is given in the *System/3 BASIC Reference Manual* in the section entitled "PRINT Statement".

The semicolon may be used as the first item in a print list causing the printhead to move 3 spaces to the right of the left margin before printing the specified print item beginning in position 4. The character constant A will print in position 4 in the following example:

```
0650 PRINT;  'A'
```

When semicolons are used in a series between print items, the printhead moves 3 spaces to the right for every semicolon after the first.

```
0950 PRINT 'AB'  ;  ;  ;  'CD'
```

Since the first semicolon in the example is preceded by a character constant, AB, the first short print zone is as long as the printed character constant. The second and third semicolons each cause the printer to move 3 spaces to the right before printing the character constant CD.

A semicolon as the last item in the print list causes printing to be continued on the same line until space is no longer available for the short print zone required by the item that is to be printed. Printing then continues on the next line.

Printing also continues on the next line when print items separated by semicolons in a print list cannot be contained in one line of print. When the line no longer has space available for the required short print zone, the printer spaces to the next line.

EXERCISE 8 - 4:

1. Enter and execute the following program segment in which the semicolon is used as the last item in the print list.

```
100  FOR I = 1 TO 100
110  PRINT  1/I;
120  NEXT I
130  END
```

2. Compare the results when this program is again executed using long precision (RUN-LONG).
3. EDIT the TABLE program previously saved.
4. Make the following modifications:
   a) Use semicolons instead of commas as separators in the print list.
   b) Omit the headings.
5. Execute the program first in SHORT precision and then in LONG precision.

   Compare these results with those printed when commas were used as separators.

DATA CONVERSION

Before results are printed, data is converted to output format. For arithmetic expressions and arithmetic variables, the printed form of a data item is determined by the precision parameter SHORT or LONG, by the number of significant digits, and by the value of the data item.

In SHORT precision, the printed value may consist of up to 6 digits, a minus sign if the value is negative, and a decimal point in the proper position if the value is not an integer.

In LONG precision, the printed value may consist of up to 11 digits, a minus sign if the value is negative, and a decimal point in the proper position if the value is not an integer.

A value is printed in E-notation (exponential form) if it exceeds the specified upper or lower limit for whole numbers and decimal numbers. In E-notation one significant digit always precedes the decimal point. This significant digit is preceded by a minus sign if the value is negative. Up to 5 digits in SHORT precision and up to 10 digits in LONG precision follow the decimal point. These digits are then followed by the letter E and assigned a one to two-digit exponent.

EXAMPLE:

1.00432E + 54
7.163E - 15

Rounding

Quantities, whether they are printed as whole numbers, decimal numbers, or in the E-notation are rounded when the number of digits exceeds the specified precision.

EXAMPLE:

Value of Expression	Print-out SHORT	LONG
-123.4560	-123.456	-123.456
-123.4565	-123.457	-123.4565
123456789	1.23457E+7	-2345678
.0999999	9.99999E-2	9.99999E-2
.09999994	9.99999E-2	9.999994E-2
.09999996	.1	9.999996E-2

In the last example, the value is represented as a decimal number in SHORT precision because through rounding, the value of the expression is equal to .1, a value equal to the lower limit for decimal numbers.

A number with a decimal point but no digits following the decimal point is not a true integer. It has a factional component that through rounding did not get printed.

EXAMPLE:

Value of expression	Printed in SHORT Precision
1.000004	1.
.9999999	1.

Section 9 Formatting Results

Performance Objectives:

- Specify a desired print format
- Specify printed numeric representation

THE IMAGE AND PRINT USING STATEMENTS

If the PRINT statement does not allow the printed format desired, you may structure the format with the IMAGE and PRINT USING statements.

These statements are used together. The IMAGE statement is a non-executable statement that describes a line to be printed. The line described is not printed until execution of the PRINT USING statement that references the IMAGE statement.

IMAGE Statement (:)

The IMAGE statement begins with a colon. The colon is followed by the description of the line to be printed. This description may be a character string, a print image, or a combination of both. In a character string, all characters of the BASIC character set except the # sign may be used:

```
110 : KEY IN THE VALUES FOR A1 AND B1
```

When this IMAGE statement is referenced by a PRINT USING statement, the characters will be printed exactly as entered.

In a print image, only the characters # + – · | may be used. The characters to be printed are provided in the PRINT USING statement that references the IMAGE statement.

```
140 : ##### ###
. . .
500 PRINT USING 140, 'GROSS', 'NET'
```

The characters 'GROSS' and 'NET' will be printed as positioned in the IMAGE statement.

In a combination of character string and print image, the characters in a character string are printed as described in the IMAGE statement but characters for the print image must be provided in the PRINT USING statement that references the IMAGE statement:

```
110 : KEY IN THE VALUES FOR ## AND ##
. . .
400 PRINT USING 110, 'A1', 'B1'
```

The character constants 'A1' and 'B1' replace the # signs in the IMAGE statement. Execution of the PRINT USING statement causes the following print out:

```
KEY IN THE VALUES FOR A1 AND B1
```

PRINT USING Statement

The PRINT USING statement consists of a line number and an optional print list. The line number in the statement must be the line number of the IMAGE statement that describes the line to be printed. The print list specifies the print items that are to replace any images in the IMAGE statement.

USING THESE STATEMENTS

Printing Decimal Values

A decimal point in the print image indicates that integer digits are to be separated from fractional digits by a printed decimal point. All # signs to the right of the decimal point are replaced by the fractional part of the value. If the number of fractional digits is less than the number of # signs right of the decimal point, trailing zeros are printed. (Note that this use of the print image forces the printing of trailing zeros, normally truncated in a print-out.) When the number of fractional digits exceeds the number of # signs to the right of the decimal point, the value is properly rounded.

All # signs left of the decimal point are replaced by the integer part of the value. If the number of integers is zero and at least one # sign is to the left of the decimal point, one zero is printed left of the decimal point. When the number of integer digits exceeds the number of # signs left of the decimal point, all asterisks are printed for that specific value.

EXAMPLE:

VALUE	PRINT IMAGE	RESULTS
12.345	##.##	12.35
123.45	####.##	123.45
123.00	####.##	123.00
000.12	###.##	0.12
12.34	#.##	****

Printing Numbers in the E-format

To have a number printed in the E-format, the print image must be followed by 4 logical OR- signs.

EXAMPLE:

#.###### ||||

When the value is printed, the 4 logical OR- signs are replaced by the letter E, a sign, and a 2-digit exponent.

The position of the decimal point in the print image determines the form of the print-out. Fractions are rounded before printing.

VALUE	PRINT IMAGE	RESULTS (In short precision)
123456.78	#.###### ||||	1.234568E+05
123456.78	###.#### ||||	123.4568E+03
123456.78	#.### ||||	1.235E+05

Using the IMAGE Statement to Print Character Values

When a print image is replaced by character items in a print using list, replacement is from left to right. If the print image is longer than the list item, the rightmost # signs are converted to blanks. If the list item is longer than the print image, the list item is truncated on the right.

```
100 PRINT USING 120 'THIS PRINT IMAGE IS NOT' 'TOO SHORT'
110 PRINT USING 130 'THIS PRINT IMAGE IS NOT' 'TOO SHORT'
120                 : ####################### #########
130                 : ###################     #########
```

PRINT OUT:

```
THIS PRINT IMAGE IS NOT TOO SHORT  (Using 120)
THIS PRINT IMAGE IS     TOO SHORT  (Using 130)
```

Multiple Use of IMAGE Statement With One PRINT USING Statement

If the number of items in the print using list exceeds the number of print images provided in the referenced IMAGE statement, the same print images are reused until the print using list is exhausted. Everytime the IMAGE statement is reused, a new line is printed.

EXAMPLE:

```
200:  ##################  ####
. . .  .
250  PRINT USING 200, 'BILL', 120, 'JOHN', 136, 'FRED', 140
```

PRINT OUT:

```
BILL    120
JOHN    136
FRED    140
```

Printing Integers in Report Format

An IMAGE statement consisting of only print images can be employed to create a table format that cannot be defined with the PRINT statement. Suppose a table of powers for N, where N has a range of values from 1 to 35, is to be printed:

N	N↑2	N↑3	N↑4
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
. . .	. . .	. . .	. . .

A table in this format cannot be produced with the PRINT statement for two reasons. First, the numbers could not be right-adjusted, that is, print in the rightmost positions of each column. Second, spacing between columns would be restricted to the spacing created by the long print zone.

With the IMAGE statement, however, the number of spaces that should occur between columns and the number of print positions in each column can be defined. To create the table of powers, the following IMAGE statement could be used:

```
100 : ##  ####  #####  ######
```

Columns are 2 spaces apart. The first column is 2 print positions wide, the second has 4 positions, the third 5, and the fourth 6.

The PRINT USING statement provides the variable name for the values to be contained in the first column and the arithmetic expressions for the values in the last 3 columns:

```
130 PRINT USING 100 N, N↑ 2, N↑ 3, N↑ 4
```

When the PRINT USING statement is executed, the assigned value for N replaces the print image ##. The value of the expression N↑2 replaces the second print image, the value of N↑3 the third print image, and the value of N↑4 the fourth print image.

Whenever a value does not require all of the positions provided in the print image, the leftmost unused positions are converted to blanks. For example, if the value for N has only one digit, the leftmost # sign is converted to a blank.

The table also requires a heading for each column. These can be created with a PRINT statement:

```
110 PRINT 'N   N↑2   N↑3   N↑4'
```

To make it easier to align the headings with the columns defined in the IMAGE statement, this PRINT statement can be entered in the line following the IMAGE statement. The quotation mark that defines the character constant in the PRINT statement should be entered directly under the colon that begins the IMAGE statement:

```
100        : ##  ####  #####  ######
110 PRINT  ' N    N↑2    N↑3    N↑4'
120 FOR N = 1 TO 35
130 PRINT USING 100, N, N↑2, N↑3, N↑4
140 NEXT N
150 END
```

EXERCISE 9 - 1:

1. Enter the preceding program to create a table of powers.
2. RUN
3. Do not clear the Work File unless you SAVE this program for later use.

   Note that when the value of the expression N↑4 exceeds 6 digits, asterisks are printed in that column. Asterisks indicate that the number of print positions required by the value is greater than the number provided in the print image.

Printing Signs

If negative values are expected, one more # sign should be added to the print image that will be replaced by the negative value. Whenever the value is negative, a minus sign will precede the printed number. If the value is positive, it will be preceded by a blank. With the following print image, a maximum of 6 positive digits or a maximum of 5 negative digits can be printed.

EXAMPLE:

```
######
 123456
    or
-12345
```

This same effect is achieved when the string of # signs in the image is preceded by a minus sign. If the value is negative, the sign is printed. When the value is positive, the printed number is preceded by a blank. The print image is the example below allows a maximum of 5 digits, positive or negative, to be printed.

EXAMPLE:

```
-#####
 12345
    or
-12345
```

If positive as well as negative signs are to be printed, the print image should begin with a + sign. Positive values will then be preceded by a + sign and negative values will be preceded by a - sign.

EXAMPLE:

```
+#####
+12345
- 12345
```

EXERCISE 9 - 2:

1. LIST the program to create a table of powers (Exercise 9 - 1).
2. Modify the program so that the values for N range from - 12 to +35.
   a) In the printed table, the values for N should be preceded by either a + or - sign.
   b) All other negative values in the table should be preceded by a - sign.
   c) Make the change that will prevent asterisks from being printed. The solution appears below.

SOLUTION:

```
100 : +## ####   ###### #######
110 FOR N = -12 TO 35
```

Since values for N↑3 will be negative, another position was printed in the third print image for the minus sign. The additional # sign position in the last print image prevents the printing of asterisks by providing an image large enough to contain the result. The + sign was added to the first image to print the sign.

Practice in Programming

Write a program that will produce a list of values for x, y, and each term of the cubic function $ax^3 - bx^2 + cx - d$. Use the following values:

$$y = 4x^3 - 12x^2 + 36x - 72$$

where the value of x ranges from -5 to +20 with an increment of 1.

EXAMPLE:

X	4*X↑3	-	12*X↑2	+	36*X	-	72	Y
-5	-500		300		-180		72	-1052
...	...		...		...		...	...
...	...		...		...		...	...
20	32000		4800		720		72	27848

This table can be used to estimate the curve of the cubic function. The number of times the curve intersects the x-axis can be determined by the values of y. This program is shown on the following page.

SOLUTION:

```
100          :###      ######  -  #####  +  ####  -  ##  +######
110 PRINT  'X          4*X↑3   -  12*X↑2  +  36*X  -  72       Y  '
120 FOR X = -5 TO 20
130 PRINT USING 100, X, 4*X↑3, 12X↑2, 36*X, 72, 4*X↑3 - 12*X↑2 + 36*X-72
140 NEXT X
150 END
```

EXERCISE 9 - 3:

1. Enter this program with the name CUFUN.
2. RUN
3. SAVE

Section 10 Functions and Subroutines

Performance Objectives:

- Identify and use System/3 BASIC functions
- Write user - created functions
- Write and use subroutines

FUNCTIONS

System Functions

The problems presented up to this point could be solved applying only relatively simple arithmetic operations expressed in a few BASIC statements. But problem-solving often demands the use of more complex functions, for instance, calculating the cotangent of an angle. Finding the value of such a function requires a series of BASIC statements.

For many frequently used mathematical functions these statements are provided as part of the BASIC programming system. Rather than writing the statements yourself, you can calculate the value of a particular function by referencing a set of statements, or *routine,* that is built into the system. Below are listed the system functions available in System/3 BASIC.

FUNCTION NAME	DEFINITION
SIN(X)	Sine of X radians
COS(X)	Cosine of X radians
TAN(X)	Tangent of X radians
COT(X)	Cotangent of X radians
SEC(X)	Secant of X radians
CSC(X)	Cosecant of X radians
ASN(X)	Arc Sine (in radians) of X
ACS(X)	Arc Cosine (in radians) of X
ATN(X)	Arc Tangent (in radians) of X
HSN(X)	Hyperbolic sine of X
HCS(X)	Hyperbolic cosine of X
HTN(X)	Hyperbolic tangent of X
DEG(X)	Convert X radians to degrees
RAD(X)	Convert X degrees to radians
EXP(X)	Natural exponent of X (e^X)
ABS(X)	Absolute value of X
LOG(X)	Logarithm of X to the base e (In X)
LTW(X)	Logarithm of X to the base 2
LGT(X)	Logarithm of X to the base 10
SQR(X)	Square root of X
RND[(X)]	Random number between 0 and 1
INT(X)	Integer part of X
SGN(X)	Sign of X defined as: if X < 0, SGN(X) = -1 if X = 0, SGN(X) = 0 if X > 0, SGN(X) = +1
DET(X)	Determinant of matrix X (X, in this case, is always a matrix name.)

A reference to a function can be made anywhere an arithmetic expression may be used. A function is referenced by its 3-character name followed by the argument in parentheses. The argument is the value for which the function is performed.

EXAMPLE:

```
100 A = 3
110 B = 4
120 PRINT SQR (A↑2 + B↑2)
130 END
```

PRINT OUT:

```
5
```

In this example, the SQR(X) function is referenced in the PRINT statement in order to calculate the square root for the argument (A↑2 + B↑2).

A complete description of BASIC System Functions can be found in the *System/3 BASIC Reference Manual.*

EXERCISE 10 - 1:

1. Enter the program shown in the example above.
2. RUN

User Functions

Additional functions may be defined with the DEF(Define) statement. A function defined in this manner is referred to as a *user function.* Once defined within a program, a user function may be referenced in the program in the same way a system function is referenced.

The DEF statement is a non-executable statement in which a function name, an arithmetic variable, and an arithmetic expression are specified:

```
100 DEF FNC(R) = (R↑ 1/2) /3
```

The function name must consist of the characters FN followed by an alphabetic character.

EXAMPLE:

```
FNA
FNR
FN$
```

The arithmetic variable in parentheses is a so-called dummy variable. When the defined function is referenced later in the program, this dummy variable is replaced by the argument of the function.

EXAMPLE:

```
100 DEF FNC(X) = (X/2)↑2*&PI
110 DEF FNS(X) = X↑2
120 FOR N = 1 TO 10
130 PRINT FNS(N), FNC(N)
140 NEXT N
150 END
```

Two functions are defined in this program: the function FNC to determine the area of a circle, and FNS the area of a square. Both functions are referenced in the PRINT statement with the variable N as the argument. The value of N replaces the dummy variable X in the DEF statements and is the value used to calculate the value of the functions. Execution of the program will create a list of areas for squares with sides ranging from 1 to 10 and circles with diameters ranging from 1 to 10.

EXERCISE 10 - 2:

Enter and execute the program shown above.

The DEF statement also allows you to define a function through reference to another function defined in the same program. An example of this appears in the program below. Statement 115 defines the function FNZ in terms of functions FNS and FNC.

```
100 DEF FNC(X) = (X/2) ↑ 2*&PI
110 DEF FNS(X) = X↑2
115 DEF FNZ(A) = FNS(A) - FNC(A)
120 FOR N = 1 TO 10
130 PRINT FNS(N), FNC(N), FNZ(N)
140 NEXT N
150 END
```

This program will produce a list of the differences in area between squares of a given side length and circles with a diameter equal to the side length as well as the areas of the squares and circles.

Not only user-defined functions but also system functions may be referenced to define another function.

```
100 B = 4
110 A = 3
120 DEF FNC(R) = -1 *SQR(R)
130 PRINT FNC ( A↑2 + B↑2)
140 END
```

The negative square root of any argument provided for the function FNC will be printed when line 130 is executed.

A function that is used to define another function is said to be nested in the function being defined. In the sample program to determine difference in areas, the function FNS and FNC are nested in the function FNZ.

Nesting also occurs if a function is used as the argument of another function:

```
100 DEF FND(X) = X↑2
110 Y = FND(FND(3))
120 PRINT Y
```

In this program the value assigned to Y is computed as follows. The constant 3 in the innermost parentheses is used as the argument for the function FND in parentheses. This value is assigned to the dummy variable X in the DEF statement and the value 9 is returned as the result of the specified function. This value is then used as the argument of the outer function. The value 9 is assigned to the dummy variable, the function is carried out, and 81 is printed as the value of Y.

SUBROUTINES

If a function is so complex that it cannot be defined with the DEF statement, a subroutine may be advantageous. A subroutine is a set of statements within a program that is executed when referenced by a GOSUB statement in the program.

The GOSUB statement references or *calls* a subroutine:

```
GOSUB 500
```

The number 500 refers to the first statement of the subroutine.

A subroutine is terminated by the RETURN statement, the last statement of the subroutine:

```
580 RETURN
```

When this statement is encountered, program execution continues with the statement following the GOSUB statement that called the subroutine.

An example will illustrate the use of subroutines. It may happen that the values of two variables have to be changed at several locations within a program. Assume that the variables A and B must exchange values. A subroutine provides a convenient method to accomplish this since a subroutine may be used several times within a program. This subroutine is shown below:

```
810 A = A + B
820 B = A - B
830 A = A - B
840 RETURN
```

Everytime the values of A and B must be exchanged in the program, a GOSUB statement referencing line 810 is used to call the subroutine.

EXAMPLE:

```
. . .
. . .
310 GOSUB 810
. . .
. . .
450 GOSUB 810
. . .
. . .
720 GOSUB 810
. . .
. . .
810 A = A + B
820 B = A - B
830 A = A - B
840 RETURN
. . .
. . .
900 END
```

Practice in Programming

Listed below is the CUFUN program presently in your library.

1. Modify the program so that the value of X has a range from -50 to +50 with an increment of 1.
2. Insert a LET statement to count the number of output lines printed.
3. Add a subroutine that will cause the printer to skip to the next page of printer paper each time 50 lines of output have been printed. (The standard printer paper - 11 inches - has 66 print lines per page.)

CUFUN Program

```
100          :###      ######   -    #####  +  ####   -  ## + ######
110 PRINT  ' X         4*X ↑3   -    12X↑2  +  36*X   -  72      Y  '
120 FOR X = -5 TO 20
130 PRINT USING 100, X, 4*X↑3, 12X↑2, 36*X, 72, 4*X↑3 - 12*X↑2 + 36*X-72
140 NEXT X
150 END
```

EXERCISE 10 - 1:

1. EDIT the saved CUFUN program.
2. Enter the program changes:
   a) Modify line 120
   b) Insert a LET statement to set-up a counter.
   c) Insert subroutine to cause paper to skip 16 spaces to new page after printing 50 lines. The subroutine should also contain a LET statement to reset the counter to zero.
3. Manually adjust the printer paper so that printing begins at the top of a page.
4. RUN
5. SAVE

Solution to Exercise 10 - 1:

CUFUN

```
100          : ###       ######  -  #####  +  ####  -  ##   +  ######
110 PRINT ‘  X        4*X↑3    -  12X↑2   +   36*X  -  72          Y  ’
120 FOR X =  -50 TO +50
130 PRINT USING 100, X, 4*X↑3, 12X↑2, 36*X, 72, 4*X↑3 - 12X↑2 + 36*X–72
140 L = L + 1
150 IF L<50 THEN 170
160 GOSUB 190
170 NEXT X
180 GO TO 240
190 FOR I = 1 TO 16
200 PRINT
210 NEXT I
220 L = 0
230 RETURN
240 END
```

Note: The GO statement (180) prevents unintentional entry to the subroutine.

Section 11 Stopping Program Execution

Performance Objectives:

Stop and continue program execution

Suspend and resume program execution

STOPPING PROGRAM EXECUTION

Program execution can be stopped in any one of the following ways:

1. normal termination by execution of the END statement.
2. interruption of execution through the INQUIRY REQUEST switch
3. termination of execution because of an execution error, for instance, INSUFFICIENT DATA FOR READ

Program execution also stops when a STOP statement or a PAUSE statement is encountered in the program.

STOP Statement

Like the END statement, the STOP statement terminates program execution and returns the system to the input status. Unlike the END statement, the STOP statement may appear anywhere in a program and may be used as often as needed within a program.

A STOP statement is normally used when more than one way of terminating execution must be provided in the program. For example, assume that if the value of an expression becomes negative, continuation of the program is meaningless. The STOP statement can be used to terminate execution.

EXAMPLE:

```
. . .
300 IF A - B >= 0 THEN 330
310 PRINT 'VALUE OF EXPRESSION NEGATIVE.  STOP'
320 STOP
330 LET. . .
. . .
. . .
. . .
900 END
```

In this example, the STOP statement terminates program execution if A - B produces a negative value. Statement 310, the PRINT statement, prints the reason for the halt. Of course, the IF. . .THEN statement could be used to branch to the END statement if the value is negative. Using the STOP statement, however, allows the insertion of a PRINT statement to explain the abnormal termination of the program.

PAUSE Statement

A PAUSE statement, like the INQUIRY REQUEST switch, puts the system in a pause status. When a PAUSE statement is executed, program execution stops and the word PAUSE followed by a line number is printed.

EXAMPLE:

```
. . .
410 Y1 = 50
420 PAUSE
430 X = Y1 + 10
. . .
. . .
RUN

PAUSE 420
```

The message "PAUSE 420" is printed when the system executes statement number 420. During a pause, the following commands may be used: SET, DISPLAY, EDIT, LIST, READ, HELP, LISTCAT, CONDITION, SYMBOLS, WIDTH, WRITE, SUSPEND, and GO. No other commands may be used in the pause status. It is not possible to modify, enter or delete a BASIC statement during the pause.

To restart execution, press the PROG START key (if no other key was pressed after the pause state was entered) or key GO and press RETURN.

Rather than continuing program execution, you can terminate the program by entering:

```
GO ABORT
```

EXERCISE 11 - 1:

1. EDIT the saved CUFUN program. (This has the subroutine to limit the output to 50 lines per page.)
2. Insert a PAUSE statement at the beginning of the program to stop execution, allowing the user to adjust the printer paper so that printing begins at the top of a page. Precede this PAUSE statement with a PRINT statement giving the operator instructions.
3. To eliminate the branch to the END statement, replace the GO TO statement with a STOP statement.
4. RUN
5. SAVE

SOLUTION:

```
 80  PRINT 'ADJUST PRINTER PAPER SO THAT'
 85  PRINT 'PRINTING BEGINS AT THE TOP OF A PAGE'
 90  PAUSE
100          :###    ######   -    #####  +  ####  -   ##  + ######
110  PRINT ' X      4*X↑3          12*X↑2      36*X  -    72        Y   '
120  FOR X = -50 TO +50
130  PRINT USING 100, X, 4*X↑3, 12*X↑2, 36*X, 72, 4*X↑3 - 12*X↑2 + 36*X - 72
140 L = L + 1
150 IF L <50 THEN 170
160  GOSUB 190
170  NEXT X
180  STOP
190  FOR I = 1 TO 16
200  PRINT
210  NEXT I
220  L = 0
230  RETURN
240  END
```

SUSPEND and RESUME Commands

The SUSPEND command can be entered when a program is in an execution pause. It causes the program to be saved so that execution can be resumed at a later time.

EXAMPLE: SUSPEND

A suspended program is different from one in an execution pause in that any system command can be entered while the program is suspended.

To continue execution of a suspended program, the RESUME command and then the GO command are entered.

```
EXAMPLE:  RESUME
          GO
```

EXERCISE 11 - 2:

1. RUN CUFUN. When the program enters the pause status, enter SUSPEND.
2. EDIT and RUN the STOCK program saved in your library.
3. Key RESUME; adjust printer paper.
4. Key GO.

Section 12 Arrays and Matrices

Performance Objectives:

- Define array dimensions
- Reference elements of arrays
- Load and print arrays
- Write matrix expressions

ARRAYS AND MATRICES

An array is an arrangement of data elements. In BASIC, arrays are of rectangular shape and may have either one or two dimensions. In a one-dimensional array (vector), elements are arranged in a single row. The row may contain from 1 to 9999 elements.

EXAMPLE: 5 8 2 1 16 10

The elements of a two-dimensional array are arranged in 2 to 9999 rows of equal length with 1 to 9999 elements per row. A two-dimensional array is also called a *matrix*.

EXAMPLE:

19	24	23
26	22	18
21	20	25

There are two types of arrays in BASIC. A *character array* contains only character data and can have only one dimension. The name of a character array consists of an alphabetic character followed by a $ sign.

EXAMPLE: A$
X$

An *arithmetic array* contains only arithmetic data and may have one or two dimensions. The name of an arithmetic array consists of an alphabetic character.

EXAMPLE: A
B

Referencing Elements of Arrays

An element of an array is referenced by the array name followed by a subscript. The subscript appears in parentheses and specifies the relative position of an element within the named array.

For a two-dimensional array, the subscript consists of two integer numbers separated by a comma. The first integer indicates the row to be referenced, the second indicates the position of the element within that row. In the following example, the fifth element in the second row of the arithmetic array A is referenced.

EXAMPLE: A(2, 5)

Arithmetic expressions, instead of integers, may be used in the subscript. In this case, the integer value of the expression is first calculated and then used to reference the element.

EXAMPLE: A(I*N, J+K)

An element in a one-dimensional array is referenced by the array name followed by a single integer in parentheses. The integer indicates the position of the element within the single row of the array.

EXAMPLE: A (5)
B$ (3)

Defining the Shape and Size of Arrays

At the beginning of a program, the maximum dimensions of the array to be processed are specified with the DIM (dimension) statement. If only one dimension is given, that dimension refers to the number of elements in the row. When two dimensions are given, the first integer refers to the number of rows and the second to the number of elements per row. In the following example, the arithmetic arrays A and L are defined as two-dimensional arrays and S is defined as one-dimensional.

```
10 DIM A (3, 5), L (110, 10), S (5)
```

In a character array, the array name is followed by a single integer only.

EXAMPLE: 10 DIM B$ (15), C$ (26)

An array that is named in a DIM statement is said to be explicitly defined. An array can also be defined implicitly.

If a reference to an array not named in a DIM statement is made in another statement, for instance a LET or a FOR statement, then the referenced array is assigned by default a maximum dimension. When the array reference has a single subscript, this maximum dimension is 10. When the array reference has a double subscript, a maximum dimension of 10, 10 is assigned to the array. However, an array may not be implicitly defined with any of the MAT statements. (MAT statements are discussed later in this section.)

The maximum dimensions of an array may be specified only once, either explicitly in a DIM statement or implicitly (be default) through a reference in another statement. If the maximum dimensions of an array are assigned by default, than an attempt to later name that array in a DIM statement will result in a compilation error.

Although the maximum dimensions of an array, as defined either explicitly or implicitly, cannot be changed during execution, the current dimensions of the array can be changed with the MAT GET, MAT INPUT, and the MAT READ statements.

Creating Arithmetic Arrays

At the start of execution all elements of all arrays dimensioned in a program are set to zero (arithmetic arrays) or blanks (character arrays). Values must be assigned to their elements, that is, the arrays have to be loaded. Some of the ways that this can be done are described on the following page.

For a sample problem, assume that a one-dimensional matrix containing the first 20 prime numbers is to be created. The prime numbers are: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71. One way to load these values is by using a series of LET statements.

EXAMPLE:

```
 10 DIM P(20)
100 P (1) = 2
110 P (2) = 3
120 P (3) = 5
130 P (4) = 7
. . .
. . .
290 P (20) = 71
```

This method is slow and requires too much effort. The job can be done more efficiently with the MAT READ statement.

MAT READ

This statement is used with the DATA statement in the same way the READ statement is used. By means of the DATA statement a series of data is entered as part of the BASIC program to create a data file. The MAT READ statement references a defined arithmetic array, and at execution causes successive elements from the data file to be read into the specified matrix. A data pointer moves over the items in the data file, always assigning the next unread item to the matrix. The matrix is loaded row by row, the value of each element being replaced by the value assigned from the data file. If the matrix requires more values than are available in the data file, an error occurs and execution is terminated.

EXAMPLE:

```
 10 DIM P(20)
100 DATA 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41
110 DATA 43, 47, 53, 59, 61, 67, 71
120 MAT READ P
. . .
```

If more than one matrix name is referenced in the MAT READ statement, the matrices are loaded one at a time, beginning with the first referenced matrix. A subscript may be included after the matrix name to redimension the matrix. If, for example, you wanted only 15 data items loaded into a matrix defined with a maximum dimension of 20, then the matrix name in the MAT READ statement would be followed by the subscript 15 to redimension the matrix.

EXAMPLE:

```
100 DIM L(20)
110 DATA 1, 2, 3, 4, . . .
120 MAT READ L(15)
. . .
. . .
```

MAT INPUT

A matrix can also be loaded by using the MAT INPUT statement.

EXAMPLE:

```
 10 DIM P(20)
100 MAT INPUT P
. . .
```

The MAT INPUT statement is like the INPUT statement. When it is executed, a question mark is printed and the keyboard is activated for input. You then enter the arithmetic data to replace the elements of the first row of the specified array.

EXAMPLE:

```
System:  ?
User:    2, 3, 5, 7, 11, . . .
```

Each data item, except the last item, is followed by a comma. If the print line is filled before all the data for the row has been entered, the last data item in the line is followed by a comma before the RETURN is pressed.

When the matrix has more than one row, two question marks are printed on the completion of a row, indicating that data can be entered for the next row.

If the number of values entered for a row does not equal the number of array elements currently dimensioned for the array row, an error message is printed requesting that the data be rekeyed for the row contining the error. This message is also printed if any of the entered values are non-arithmetic or invalid (too great or too small).

More than one matrix can be referenced with the MAT IMPUT statement, and the referenced matrices may be redimensioned by including the optional subscripts.

Creating Character Arrays

MAT statements are not used with character arrays. These are loaded using either an appropriate number of LET statements or, better, a FOR loop.

EXAMPLE:

```
 10 DIM C$(32)
100 DATA 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M','N', 'O', 'P', 'Q'
110 DATA 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'
120 FOR I = 1 TO 26
130 READ C$ (I)
140 NEXT I
. . .
. . .
```

The program above creates a character array consisting of all alphabetic characters A to Z. The values that will replace the array elements are entered with the DATA statement. To load the array with these values, the READ statement is used. Unlike the MAT READ statement which causes the data items to be assigned to consecutive elements of the array, the READ statement requires a reference to the element to which the data item will be assigned. This reference is the subscript I, which is also the control variable for the FOR loop. The FOR loop will be executed 26 times in order to read the 26 data items in the data file. Since the control variable is the same as the subscript, the data item read during the first execution of the loop is assigned to the first element of the array; the data item read the second time the loop is executed is assigned to the second element of the array, and so on.

On completion of the loop, only 26 elements of the 32-element array will be loaded. These remaining elements could be loaded later in the program if required.

No execution error occurs if all elements of the array are not assigned values. However, execution terminates if an attempt is made to assign data items to the array when there is insufficient data in the data file.

Printing Matrices

Entire arithmetic arrays can be printed with the MAT PRINT or the MAT PRINT USING statements. These statements are similar to the PRINT and PRINT USING statements.

On execution of the MAT PRINT statement, each referenced matrix is printed row by row, one row per line with a single blank line separating the rows. Each matrix begins on a new line and is separated from the preceding matrix by two blank lines.

Commas and semicolons in the mat print list cause horizontal spacing. If a matrix name in the mat print list is followed by a comma, the elements are printed in long print zones. If followed by a semicolon, the elements are printed in short print zones. Below are examples of MAT PRINT statements:

```
700 MAT PRINT A          (elements of arrays A, B, C, and D are printed
800 MAT PRINT B,          in long print zones, one row per line)
900 MAT PRINT C, D

950 MAT PRINT E;         (elements of arrays E, X, and Z are printed
960 MAT PRINT X; Z;      in short print zones, one row per line)
```

EXERCISE 12 - 1:

Write a program to create an array N (9,4) that contains the numbers 1 to 36.

a) Enter the numbers with the MAT INPUT statement.

b) Print the array in long print zones.

This program is shown on the following page.

Program to create the print an array containing the numbers 1 to 36:

```
 90 DIM N(9, 4)
100 MAT INPUT N
110 MAT PRINT N
120 END
```

MAT PRINT USING

Like the PRINT USING statement, the MAT PRINT USING statement works in conjunction with an IMAGE statement. The MAT PRINT USING statement references the IMAGE statement that describes the format in which the rows of the specified matrix or matrices will be printed.

Each matrix row is printed on a new line and is separated from the preceding row by a single blank line. If there are more elements in a matrix row that print images in the IMAGE statement, the IMAGE statement is reused and printing of that matrix row is continued on the next line. If at the end of a row unused print images are left in the IMAGE statement, that print line is terminated at the first unused print image.

EXAMPLE:

```
100 DIM P(4, 5)
110 DATA 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41
120 DATA 43, 47, 53, 59, 61, 67, 71
130 MAT READ P
140 : SOME PRIME NUMBERS ARE ## ## ## ## ##
150 MAT PRINT USING 140, P
160 END
```

Prints both character string and matrix values on each line. Could use a PRINT statement to print character string once, and the Image statement to print the 4 rows of the matrix.

PRINT OUT:

```
SOME PRIME NUMBERS ARE   2   3   5   7  11
SOME PRIME NUMBERS ARE  13  17  19  23  29
SOME PRIME NUMBERS ARE  31  37  41  43  47
SOME PRIME NUMBERS ARE  53  59  61  67  71
```

Printing Character Arrays

The MAT PRINT and the MAT PRINT USING statements will print only matrices. To print character arrays, a PRINT or PRINT USING statement is employed.

EXAMPLE:

```
 90 DIM C$ (32)
100 DATA 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
110 DATA 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'
120 FOR I = 1 TO 26
130 READ C$ (I)
140 NEXT I
. . .
. . .
800 FOR I = 1 TO 26
810 PRINT C$ (I);;
820 NEXT I
830 END
```

This is the program to create the character array containing the alphabet. Another loop has been added to get the array printed. Because the last item in the PRINT statement is a semicolon, elements will be printed in the same print line, each within a short print zone and separated from each other by a short print zone.

MAT Assignment Statement

In a MAT statement, the values in the matrix to the right of the = sign are assigned to the matrix named on the left of the= sign. The matrices must have the same dimensions.

```
EXAMPLE:
100 DIM A (10, 10), B (10, 10)
. . .
. . .
. . .
150 MAT A = B
```

If a matrix expression is specified to the right of the = sign, the operations in the expression are performed before the values are assigned to the matrix on the left.

Valid matrix expressions and the operations they perform are shown below. In these expressions M and N represent matrices and the e, represents an arithmetic constant, an arithmetic expression, or an arithmetic variable.

M + N Sum of two matrices

```
EXAMPLE:    MAT A = A + E
```

The dimensions specified for the matrix to the left of the = sign must be the same as the dimensions of the matrices in the matrix expression.

M – N Difference of two matrices

```
EXAMPLE:    MAT A = C – D
```

As in matrix addition, the dimensions of the matrix to the left of the = sign must be the same as the dimensions of the matrices in the matrix expression.

M * N Product of two matrices according to mathematical matrix multiplication

```
EXAMPLE:    MAT C = A * B
```

In matrix multiplication the total size of the matrix to the left of the = sign must be the same size as the product matrix. Only two-dimensional matrices can be specified in matrix multiplication. The matrix named to the left of the = sign must not appear on the expression to the right.

(e) * M — Product of e (arithmetic constant, expression, or variable enclosed in parentheses) and matrix M

EXAMPLE: MAT C = (10) * R

The size of the matrix to the left of the = sign must be the same as the size of the matrix named on the right. Either one or two-dimensional matrices may be specified.

ZER (e_1,e_2) — Produces an e_1 or e_1 -by- e_2 zero matrix, that is, a matrix in which all elements assume the value zero (0).

EXAMPLE: MAT A = ZER
MAT B = ZER (A + D, R – S)

By including the subscript, the matrix will be redimensioned. However, this matrix size cannot exceed the maximum size of the matrix as defined either implicitly or explicitly.

CON (e_1,e_2) — Produces an e_1 or e_1 -by- e_2 matrix, that is, a matrix in which all elements have a value of one((1).

EXAMPLE: MAT A = CON
MAT B = CON (R, S)

The matrix can be redimensioned by including subscripts but this new size cannot exceed the maximum size of the matrix as specified either implicitly or explicitly.

IDN (e_1,e_2) — Produces an e_1 -by- e_2 identity matrix. A two-dimensional square matrix must be used.

EXAMPLE: MAT A = IDN
MAT B = IDN (4, 4)

Subscripts can be used to redimension the matrix but the maximum size of the matrix must not be exceeded.

INV (M) — Inverse of matrix M. The dimensions of matrix M should not exceed 30, 30. A two-dimensional square matrix must be used.

EXAMPLE: MAT A = INV (B)

In this example matrix A is replaced by the inverse of matrix B. The matrix named to the left of the = sign must not appear in the expression to the right.

TRN (M) Transpose of matrix M. The matrix must be two-dimensional.

EXAMPLE: MAT A = TRN (B)

Matrix A in the example is replaced by the transpose of matrix B. The dimensions of the matrix to the left of the = sign must be the reverse of the matrix dimensions in the matrix expression. The matrix named to the left of the = sign must not appear in the expression to the right.

Practice in Programming

Five different materials are used to produce a product. The amounts of each material vary with the product model: economy, super, deluxe. The table below shows the number of units of each material required for each model.

	Material #1	Material #2	Material #3	Material #4	Material #5
Economy	2	4	2	3	2
Super	3	5	4	4	2
Deluxe	5	6	8	7	4

Given in the following table is the purchase cost per unit and the transportation cost per unit for each material.

	Purchase Cost / Unit	Transportation Cost / Unit
Material #1	10	5
Material #2	6	7
Material #3	5	6
Material #4	2	4
Material #5	4	2

Write a program that will calculate for each product model the purchase cost and the transportation cost of the required materials. The printed result should be a 3-row, 2-column table with the following headings: PURCHASE COST, TRANSPORTATION COST. The values in each row should be identified as costs for either the ECONOMY, SUPER, or DELUXE model. Then find the total production cost for each model by adding the purchase cost and transportation cost. A second printed result should list these totals. This second print-out should have the headings ECONOMY, SUPER, DELUXE, and the single row of values should be preceded by character constant TOTAL COST.

The array containing the quantities should be called A, the array with the unit costs should have the name B, and the product of these arrays should be the array C. In matrix multiplication, the number of columns in one matrix must equal the number of rows in the second matrix. The two tables in the problem meet this requirement.

Use the DATA statement to enter the data, and the MAT READ statement to load the arrays. Matrix multiplication is accomplished with the MAT assignment statement.

To find the total production cost for each model, the two elements in each row of the product array C must be added. Each element that is to be added must be referenced by the array name followed by a subscript.

```
EXAMPLE:
110 PRINT C(1, 1) + C(1, 2)
120 PRINT C(2, 1) + C(2, 1)
130 PRINT C(3,1) + C(3, 1)
          or
110 FOR R = 1 TO 3
120 FOR

!!)

110 FOR R = 1 TO 3
120 PRINT C(R, 1) + C(R, 2)
130 NEXT R
```

The solution follows, but try writing this program yourself before looking at the solution.

Program to compute purchase cost, transportation cost, and total cost:

```
100 DIM A(3, 5), B(5, 2), C(3, 2)
110 DATA 2, 4, 2, 3, 2, 3, 5, 4, 4, 2, 5, 6, 8, 7, 4, 10, 5, 6, 7, 5, 6
120 DATA 2, 4, 4, 2, 'ECONOMY', 'SUPER', 'DELUXE'
130        : #######      ####              ####
140 PRINT '             PURCHASE COST   TRANSPORTATION COST '
150 MAT READ A, B
160 MAT C = A * B
170 FOR I = 1 TO 3
180 READ M$
190 PRINT USING 130, M$, C(I, 1), C(I, 2)
200 NEXT I
210 PRINT, 'ECONOMY', 'SUPER', 'DELUXE'
220 PRINT 'TOTAL COST',
230 FOR F = 1 TO 3
240 PRINT C(F, 1) + C(F, 2),
250 NEXT F
260 END
```

EXERCISE 12 - 2:

1. Enter the above program.
2. RUN

Section 13 File Processing

Performance Objectives:

- Create a keyboard - generated data file
- Use a saved data file as input to a program
- Create a program - generated data file
- Update a data file

CREATING DATA FILES

Data files may be created as part of a program using the DATA statement or independent of a program. Data files that are independent can be saved in your library and then used as input data for various programs. In this way, large amounts of data can be stored without using space within a program. Also, data that is common to a number of programs need be entered only once.

This type of data file is defined with the EDIT command. Along with the command you must specify the name of the data file followed by a comma and the word DATA.

EXAMPLE:

```
EDIT INVEN, DATA
```

If the numeric values that will be entered into the file are to be stored in long precision, then the word LONG preceded by a comma should also be specified.

EXAMPLE:

```
EDIT INVEN, DATA, LONG
```

Otherwise all numeric values contained in the data file will be in short precision, regardless of the precision in which they are entered. Character constants stored in the file have an overall length of 18 characters; any characters entered beyond the 18 limit are ignored.

The data items placed in the file may be arithmetic constants, character constants, or both. They are entered in lines, each line beginning with a line number that is followed by at least one blank. No blanks may occur in the line number. To generate the line number, the PROG START key can be used. Data items are separated from each other by either a comma or a blank.

EXAMPLE:

```
100 17, 35, 84, 96, 'TOTAL', 13, 26, 41
110 52, 'TOTAL'
```

or

```
100 17 35 84 69 'TOTAL' 13 26 41
110 52 'TOTAL'
```

As each line is entered, it undergoes a syntax check. The error correction procedures described for program lines also apply to data file lines.

Regardless of the line structure, however, a data file is considered a string of data items. In this respect, the data file created with the EDIT command is like the file created with the DATA statement.

A data file entered into the Work File is placed in the library with the SAVE command.

EXAMPLE:

```
SAVE
```

A saved data file can be copied into the Work File with the EDIT command followed by the filename. The parameter DATA is not specified.

EXAMPLE: EDIT INVEN

It can be listed with the LIST command and, if necessary, modified.

EXERCISE 13 - 1:

Given below is a file showing unit cost, on-hand quantity, and annual use of specified inventory items. (Unique data -0, 0, 0, 0- is included to indicate end-of-file.)

Item No.	Unit Cost	On-Hand Qty.	Annual Use
1248	10.00	15	155.28
1765	21.50	8	207.46
1920	19.75	5	90.05
2357	12.58	6	52.25
2891	9.46	20	330.60
0	0	0	0

1. Place these data items in a file called INVEN. Use SHORT precision. Do not include the headings. Read the above file left to right, that is, enter an item number constant followed by a unit cost value, then an on-hand value, and finally an annual use value. Since the file is regarded as a string of data items, the number of items entered per line is important only with respect to the readability of the file when the LIST command is entered. Remember, each line must begin with a line number.
2. SAVE the file.

USING A SAVED DATA FILE

To use a saved data file as input for a program, the data file must first be associated with the program. Then, the program must reference the data file.

A data file is associated with a program by the ALLOCATE command. In the command, the name under which the data file is saved and, in parentheses, the name by which the data file will be referenced in the program are specified.

EXAMPLE: ALLOCATE INVEN (ORDER)

This ALLOCATE command indicates that the data file saved under the name INVEN will be referenced by the name ORDER in the program.

The program with which the file becomes associated is the program that is in the Work File. Before the ALLOCATE command can be entered, the Work File must have been cleared with the EDIT command and given the name of the program that will reference the data file.

EXAMPLE:

```
EDIT EOQ
ALLOCATE INVEN (ORDER)
```

Here the data file called INVEN becomes associated with the program called EOQ that is in the Work File. The program EOQ will use the name ORDER to reference the data file. It is not necessary that the program lines actually be in the Work File at the time the ALLOCATE command is entered.

In the program, the reference to the allocated data file is made with the GET statement.

EXAMPLE:

```
. . .
200 GET 'ORDER', I, U, Q, A
. . .
```

When the GET statement is executed, the referenced file is opened and values in the file are read sequentially and assigned to the variables specified in the statement. In the example above, values from the saved data file INVEN, referenced by the name 'ORDER', are assigned to the variables I, U, Q, and A. The variables are assigned values in the order that they appear in the statement.

As each data item in the file is assigned, a data pointer, originally set to the first item in the file, moves to the next data item in the file. When all the variables specified in the GET statement have been assigned values, the data pointer is set to the next data item in the file and reading stops. If a GET statement that references the same file is later executed, data items are read beginning at the location indicated by the data pointer.

The variables specified in the GET statement must be the same type of data (arithmetic or character) as the corresponding data items in the data file or an error occurs. Execution terminates if the number of variables in the GET statement exceeds the number of data items in the file.

Saving a Program Associated with a File

If a program to which a file has been allocated is saved, its association with the file is also saved. When the program is reused, an ALLOCATE command does not have to be entered.

Practice in Programming

One of the problems of inventory control is to determine how much to order. As a guide for deriving the order quantity, the Economic Order Quantity (EOQ) formula can be applied.

$$Q = \sqrt{\frac{2AS}{IC}}$$

Q = order quantity
A = Annual usage
S = order cost
I = item unit cost
C = inventory carrying rate

Write a program that references the saved data file INVEN to solve the inventory control problem described above. Reference the file by the name ORDER.

For each item in the INVEN file, determine the order quantity. Print the item number (N) and the order quantity (Q).

Values for the variables A and I are in the data file. For S, use the constant 10.00, and for C, the constant .20. Note that the values for the On-Hand Quantity, stored in the data file, are not required. However, you cannot prevent these values from being read.

To prevent the end-of-file data (all zeros) from being used in the calculations and causing a division-by-zero error, use an IF. . .THEN statement to test for the end of the file.

The program is shown below.

```
EOQ Program
100 S = 10.00
110 C = .20
120 PRINT 'ITEM NO.', 'ORDER QUANTITY'
130 GET 'ORDER', N, I, H, A
140 IF N = 0 THEN 170
150 PRINT N, SQR (2*(A*S)/(I*C))
160 GO TO 130
170 END
```

EXERCISE 13 - 2:

1. EDIT EOQ
2. ALLOCATE INVEN
3. Enter EOQ program
4. RUN
5. SAVE

PROGRAM - GENERATED DATA FILES

Instead of defining a data file with the EDIT command and entering the data items, data files may be created as output of a program and saved in your library. These data files can be referenced as input data for other programs. This type of file is referred to as a program-generated data file in contrast to the data file created with the EDIT command, which is keyboard-generated.

To create a program-generated data file:

1. EDIT the Work File with the name of the program that will generate the data file.
2. Name the data file that will be created and reserve space for the file in your library. This is done with the ALLOCATE command.
3. In the program, reference the data file in which the output from the program will be placed and specify the output you want saved. This reference must be made in a PUT statement.

Suppose you wanted the output from the program called TABLE to be saved in a data file. TABLE is the program, now in your library, that produces for specified values the reciprocal value, the square of the value, the circumference of a circle with the value as the diameter, and the area of a circle with the value as the diameter.

```
TABLE
0100 PRINT 'N', '1/N', 'N↑2', 'N*&PI', '(N/2)↑2*&PI'
0110 PRINT
0120 FOR N =1 TO 35
0130 PRINT N, 1/N, N↑2, N*&PI, (N/2)↑2*&PI
0140 NEXT N
0200 END
```

To place the output from this program in a data file, first EDIT the program.

EXAMPLE: EDIT TABLE

Since TABLE is saved in your library, the program lines will be copied to the Work File. The program, however, must be modified to include the PUT statement.

The PUT statement references the data file that will contain the program output and specifies the data that will be placed in the file.

EXAMPLE:

```
0130 PUT 'NUMS', N, 1/N, N↑2, N*&PI, (N/2)↑2*&PI
```

This PUT statement references the file that will be created by the name NUMS. The expressions that follow this reference name indicate the values that will go into the file. In the TABLE program, the PUT statement must be inserted before the NEXT statement, that is, before the value of N is changed.

```
0120 FOR N = 1 TO 35
0130 PUT 'NUMS', N, 1/N, N↑2, N*&PI, (N/2)↑2*&PI
0140 NEXT N
0200 END
```

If it is not desirable to have the output printed, the PRINT statements can be deleted from the program. The expressions in the PUT statement will be calculated when the PUT statement is executed. Just as the expressions in LET and PRINT statements are evaluated before they are assigned or printed, so in the PUT statement the expressions are evaluated before they are placed in the data file.

The program cannot be executed, however, until an ALLOCATE command has been entered. This command specifies:

a. The library filename of the data file that will be created. This is the name under which the file will be saved in the library.

b. The reference filename of the file that will be created. This is the name by which the program used to generate the file will reference the file and is the filename in the PUT statement.

c. The word NEW. This indicates that a new data file is to be placed in your library; space in your library is then made available. No SAVE command has to be entered.

EXAMPLE:

ALLOCATE VALUS (NUMS), NEW

In the example, the ALLOCATE command specifies that a new data file, which is to be saved under the name VALUS, is going to be created. Also, it indicates that the program which will generate this file will reference it by the name NUMS in the PUT statement. If the numeric values to go into the file are to be in long precision, then the parameter LONG must also be specified. Otherwise all numeric values will be short precision.

EXAMPLE:

ALLOCATE VALUE (NUMS), NEW, LONG

After the ALLOCATE command has been entered, the program in the Work File can be RUN. When the PUT statement is executed, the file being created is opened and a data pointer is set to the first file location. The calculated value of the first expression in the PUT statement is placed in this first file location, and the data pointer moves to the next file location, where the calculated value of the second expression specified in the PUT statement is placed. Values for the expressions are put in the file in the order that they appear in the PUT statement.

At termination of the program, the file that has been generated is in your library available as input to any program. To check the contents of this program-generated file, EDIT the file into the Work File and then LIST it.

EXAMPLE:

```
EDIT VALUS
LIST
```

A program-generated data file has no line numbers and cannot be modified with the correction procedures used to change data in the Work File. A program is required to read and modify the file.

EXERCISE 13 - 3:

1. EDIT the saved program TABLE.
2. Delete the PRINT statement that causes line spacing.
3. Replace the second PRINT statement with the following PUT statement:
   130 PUT 'NUMS', N, 1/N, N↑2, N*&PI, (N/2)↑2*&PI
4. ALLOCATE a file to be generated by this program. The file should have the library filename VALUS and the reference filename NUMS. The values placed in this file should be in LONG precision.
5. RUN
6. EDIT VALUS and then LIST.

Practice in Programming

Using the data file INVEN and the program EOQ, create another data file to be called INVEN2. In addition to the data already contained in INVEN, the file INVEN2 should contain the order quantity for each item. It is not necessary to print the output.

Since INVEN is already associated with EOQ, only one ALLOCATE command is required. This command will reserve space in your library for the data file to be generated, that is, INVEN2. In the program, reference INVEN2 with the name BUY.

EXERCISE 13 - 4:

1. EDIT EOQ and modify to create the new file INVEN2.
2. ALLOCATE INVEN2.
3. RUN
4. EDIT INVEN2 and then LIST.

Solution to 13 - 4:

```
ALLOCATE INVEN2 (BUY), NEW
100 S = 10.00
110 C = .20
120 GET 'ORDER', N, I, H, A
130 IF N=0  THEN 170 <------ { prevents zeros in INVEN file from being
                               calculated, resulting in a division by 0 error.
140 PUT 'BUY', N, I, H, A, SQR(2*(A*S)/(I*C))
150 GO TO 120
160 PUT 'BUY', N, I, H, A, X <------ { places 0, 0, 0, 0, 0 as last data items
                                       in INVEN2
170 END
```

MAT GET Statement

The MAT GET statement, like the GET statement, reads data from saved data files that have been associated with the program by the ALLOCATE command. It then places the data in arithmetic arrays (matrices). All data read from the data file must be arithmetic or an error occurs.

The matrices specified in the statement must have been previously defined in the program, either implicitly or explicitly. However, the specified matrices can be redimensioned with the MAT GET statement by including subscripts after the matrix name.

EXAMPLE:

```
100 DIM X(4, 3), Y(5, 5)
. . .
200 MAT GET 'ITEMS', X, 'FILE', Y(3,3)
```

When the MAT GET statement is executed the data file is opened, a data pointer is set to the first location in the file, and the data items are read sequentially, the data pointer always moving ahead to the next data item. The data is loaded into each specified matrix by row in the order that the matrices are listed in the MAT GET statement.

If the number of data items in the data file is less than the number of items necessary to fill all the matrices indicated in the MAT GET statement, execution is terminated. When all the matrices specified have been filled, the data pointer is set to the next data item in the file and reading stops. Execution of a subsequent GET or MAT GET statement that references the same file will begin reading at the location indicated by this data pointer.

In the EOQ program, the MAT GET statement can be used to read data from the saved INVEN data file. To place the data from the file into a matrix, a DIM statement must be added to the program to define the matrix. Since the data in the file is divided into groups of four (item no., unit cost, on-hand quantity, and annual usage) and there are 6 such data groups, a 4-column, 6-row matrix is defined. The matrix is arbitrarily called A.

EXAMPLE: `90 DIM A(6, 4)`

The GET statement must be changed to a MAT GET statement that references the file and specifies the matrix into which the data items will be loaded.

```
EXAMPLE:    GET 'ORDER', A
            MAT GET 'ORDER', A
```

This modified EOQ program is shown below.

```
 90 DIM A(6, 4)
100 S = 10.00
110 C = .20
120 MAT GET 'ORDER', A
140 PRINT 'ITEM NO.', 'ORDER QUANTITY'
145 FOR X = 1 TO 5
150 PRINT A (X,1), SQR(2*(A(X,4)*S)/(A(X,2)*C))
160 NEXT X
165 PRINT A(6,1),A(6,1) <-------(causes the end-of-file data to be printed
170 END
```

(causes the item no. of each data group to be printed and the EOQ for each data group to be calculated and printed)

If the EOQ program now in the library were copied to the Work File and modified as shown above, an ALLOCATE command would NOT have to be entered. The data file INVEN is already associated with the EOQ program.

MAT PUT Statement

When the result of a program is a matrix, the output is placed in a program-generated data file with the MAT PUT statement. The matrices specified in the statement must have been previously defined in the program, either implicitly or explicitly. The reference filename must be the same reference filename used in the ALLOCATE command that reserves library space for the data file.

```
EXAMPLE:    100 DIM M(4, 5)
            . . .
            700 MAT PUT 'NUMS', M
```

When the MAT PUT statement is executed the file is opened and row after row, the data items from the specified matrices are placed in the data file. As each data item is placed in the file, a data pointer, originally set to the first file location, moves ahead to indicate the file location for the next data item.

When all of the data in the specified matrices have been placed in the data file, the pointer is set to the next available data location. Execution of a subsequent PUT or MAT PUT statement that references the same data file will place data in the file at the location indicated by the data pointer.

The following program places the numbers 1 to 25 in the program-generated file called TESTDATA, referenced in the program by the name DIGITS:

```
ALLOCATE TEST DATA (DIGITS), NEW
 90 DIM A(5, 5)
100 DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
110 DATA 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25
120 MAT READ A
130 MAT PRINT A
140 MAT PUT 'DIGITS', A
150 END
```

UPDATING FILES

To change values in a data file created with the EDIT command, you simply copy the file into the Work File, modify those lines in which you want the values changed, and save the file.

But these procedures for modifying a file do not apply to program-generated data files, for instance, INVEN2. Program-generated files do not have line numbers. These files you modify, or update, with a program. In the program, values are read from the file and assigned to variables. Then the values of these variables can be changed.

```
EXAMPLE:
ALLOCATE NUFILE (DATA)
100 GET 'DATA', A, B, C, D
110 PRINT A; B; C; D
120 LET A = B + 2
130 LET C = D - 2
140 PRINT A; B; C; D
```

After the values have been read, the file must be closed. This is done with the CLOSE statement. The CLOSE statement moves the data pointer back to the first location of the data file. The data file can then be referenced again as either an input file or an output file. Only data files that have been opened with a PUT, MAT PUT, GET, or MAT GET statement can be closed with the CLOSE statement.

```
150 CLOSE 'DATA'
```

With the file closed, it can be referenced as an output file for the values assigned to the variables in the program.

```
160 PUT 'DATA', A, B, C, D
170 END
```

The specified data items are placed in the file beginning at the first file location. As each data item is put in the file, it replaces the item stored in the file location. The last data item specified in the PUT statement becomes the last data item in the file. If, for instance, a data file consisting of 50 data items is read but only 45 are put back in the file, the 45th data item becomes the last data item in the file. Even though data items 46 through 50 are not replaced by new values, they can no longer be accessed by a GET or MAT GET statement.

Before the preceding program can be executed, one ALLOCATE command must be entered. In the command the library file name of the data file and the name by which both the GET statement and the PUT statement will reference the file are specified.

EXAMPLE:
ALLOCATE NUFILE (DATA)

Practice in Programming

Given below is a data file PRAC consisting of 10 values.

PRAC
25, 60, 42, 71, 3, 19, 56, 88, 67, 9

Write a program to change the fourth value to 36 and the eighth value to 29. Reference the file with the name TEST. This program is shown below.

```
100 GET 'TEST' , A, B, C, D, E, F, G, H, I, J
110 PRINT A; B; C; D; E; F; G; H; I; J
120 LET D = 36
130 LET H = 29
140 CLOSE 'TEST'
150 PRINT A; B; C; D; E; F; G; H; I; J
160 PUT 'TEST', A, B, C, D, E, F, G, H, I, J
```

EXERCISE 13 - 5:

1. Create the data file PRAC with the EDIT command. Remember to enter a line number.
2. SAVE PRAC
3. EDIT and enter the program shown above.
4. ALLOCATE PRAC
5. RUN
6. EDIT PRAC and then LIST.

Often a data file consists of groups of data. The INVEN files, in which an item number is entered followed by relevant data and then another item number, is a case in point. Updating these kinds of files usually involves changing one or two data items within each data group.

In the program that updates this kind of file, the data is read, modified, and placed back in the file in groups.

EXAMPLE:

```
ALLOCATE DATAFIL(IN)
ALLOCATE DATAFIL(OUT)
110 GET 'IN', A, B, C, D, E
120 C = C + E
130 PUT 'OUT', A, B, C, D
140 GO TO 110
150 END
```

In this program, the value of the third item in each data group is changed and the fifth item in each group is deleted from the file.

Because only one data group is read at a time, the file cannot be closed with the CLOSE statement before the modified data is placed in the file. Instead, two data pointers are used at the same time, one to read data from the file, the other to place data in the file. The read data pointer must always be ahead of the data pointer used to place data back in the file.

Since two data pointers are operating at the same time, two reference filenames (and therefore two ALLOCATE commands) are required. One reference filename is associated with the GET data pointer; the other with the PUT data pointer.

When the program terminates, fewer data items have been placed in the file than were originally read from the file. Groups of 5 were read but groups of 4 were placed in the file. The file now consists of all those data items placed in the file with the PUT statement. Those data items that were not replaced by different values are no longer accessible.

Practice in Programming

Write a program that will update the data file INVEN2 by adding the 5th data item of each data group to the 3rd data item of each group. The 5th data item should then be deleted. Use the GET and the PUT statements. Then rewrite the program using the MAT GET and MAT PUT statements. Reference the file by the names IN and OUT.

EXERCISE 13 - 6:

1. EDIT and enter one of your UPDAT programs.
2. ALLOCATE INVEN2
3. RUN
4. EDIT INVEN2 and LIST

SOLUTION to 13 - 6 Using GET and PUT

```
ALLOCATE INVEN2(IN)
ALLOCATE INVEN2(OUT)


100 GET 'IN', A, B, C, D, E
110 IF A = 0 THEN 150
120 C = C + E
130 PUT 'OUT', A, B, C, D
140 GO TO 100
150 PUT 'OUT', A, B, C, D
160 END
```

or

SOLUTION to 13 - 6 Using MAT GET and MAT PUT

```
100 DIM A(4)
110 MAT GET 'IN', A
120 IF A(1) = 0 THEN 170
130 GET 'IN', X
140 A (3) = A (3) + X
150 MAT PUT 'OUT', A
160 GO TO 110
170 MAT PUT 'OUT', A
180 END
```

RESET Statement

This statement returns the data pointer to the first data item of a specified data file previously referenced by a GET, MAT GET, PUT, or MAT PUT statement.

```
310 RESET 'SUM'
```

If the file was previously referenced by a GET or MAT GET statement, then the file can again be used as input.

EXAMPLE:

```
ALLOCATE MAT 'SUM'
100 GET 'SUM', X, Y$, Z
110 RESET 'SUM'
120 GET 'SUM', A, B$, C
130  PRINT X; Y$;; Z; A; B$;; C
140 END
```

PRINT OUT

```
456   TOTAL   90   456   TOTAL   90
```

If the file was previously referenced by a PUT or MAT PUT statement, then the reset file can again be used for output.

```
EXAMPLE:
ALLOCATE POWERS (CURRENT), NEW
100 FOR C = 1 TO 10
110 PRINT C↑2
120 PUT 'CURRENT', C↑2
130 NEXT C
140 RESET 'CURRENT'
150 FOR C = 1 TO 10
160 PRINT C↑3
170 PUT 'CURRENT', C↑3
180 END
```

In this program, execution of the first FOR. . .NEXT loop places the square of values 1 to 10 in the referenced file. When the second loop is executed, the values in the referenced file are replaced by the cube of the values 1 to 10.

Section 14 Debugging

Performance Objectives:

Identify and correct program errors

DEBUGGING

A program may execute successfully yet not produce the results required. In this case a mistake has been made in the design of the program. The process of finding and removing these mistakes, or "bugs", is called debugging.

TRACE Command

One method of debugging is to trace the values of data as they are changed during program execution. To trace values, enter the TRACE command instead of the RUN command. The program is executed and the new value of each variable in the program is printed each time it is changed. If only certain variables are to be traced, the command is entered with the names of these variables.

```
TRACE X, Y
```

If arrays are to be traced, the array variables must be specified.

```
TRACE  A(3), B(5, 2), A$(4)
```

An asterisk enclosed in parentheses following an array variable indicates that the contents of the whole array are to be traced.

```
TRACE A(*), B(*), A$(*)
```

The program below, which attempts to average every 3 data items read from the data file, contains an error that might not be obvious from reading the program statements.

```
100  REM THIS PROGRAM HAS AN ERROR
110  DATA 50, 75, 100, 10, 20, 30, 5, 10, 15
120  FOR I = 1 TO 3
130  READ S
140  T = T + S
150  NEXT I
160  PRINT 'AVG IS' T/3
170  GO TO 120
180  END
RUN

PRINTED RESULT
AVG IS 75
AVG IS 95
AVG IS 105
ERROR 721 INSUFFICIENT DATA FOR READ
```

However, the second and third printed results are obviously wrong.. The average of the 4th, 5th, and 6th data items should be 20, and the average of the last group of data items should be 10. Apparently, the total that is averaged the second and the third time is greater than it should be. Executing this program again with a TRACE command in which the variable T is specified (TRACE T) produces the following information:

```
T = 50
T = 125
T = 225
AVG IS 75
T = 235
T = 255
T = 285
AVG IS 95
T = 290
T = 300
T = 315
AVG IS 105
```

The 4th, 5th, and 6th data items have been added to the total of the first 3 data items, and the last 3 data items have been added to the total of the first 6 data items. To correct the error, a statement which resets the value of T to 0 after 3 data items have been totaled and averaged must be added to the program.

Practice in Programming

The program below attempts to calculate interest on $1 for 5 years, first when the interest is 2%, again at 3%, and finally at 5%. The program has two errors.

```
100 C = 1
110 DATA 1.02, 1.03, 1.05
120 READ I
130 C = C * I
140 T = T + 1
150 IF T < 5  THEN 130
160 PRINT 'AFTER 5 YEARS AT' I 'INTEREST CAPITAL IS' C
170 GO TO 120
```

The error can be found by tracing variables.

EXERCISE 14 - 1:

1. Enter and RUN the ERROR program.
2. Execute again by entering:
   TRACE C, T

To correct the errors in the program, a statement that sets T to 0 after the interest has been calculated for 5 years must be added. Also, the GO TO statement in line 170 must be changed so that the program is branched back to line 100 where the value of C is reset to $1.

The sequence in which program statements are executed during program execution may also be traced. If FLOW is specified with the command, the line number of each statement is printed as it is executed.

```
TRACE FLOW
```

To trace variables as well as flow, the word ALL or the variable names are specified also.

```
TRACE FLOW, ALL
TRACE ALL, FLOW
TRACE FLOW, A, B, C$
```

Any array variables that are to be traced must be specified.

```
TRACE FLOW, A(2, 2), B(*)
```

Practice in Programming

The program below is to compute the roots of the following quadratic equations:

$$14x^2 - 2x = 18.6$$

$$-12x^2 + 4x = 36$$

To solve, the equation is arranged in the form:

$$ax^2 + bx + c = 0$$

and the formula for finding the roots of a quadratic equation is applied:

$$\text{roots} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

In the program, the sign of the quantity $b^2 - 4ac$ (called the discriminant) is tested, If the sign is negative, the program is to branch to the PRINT statement that prints a message indicating the roots are complex. Then the complex roots are to be calculated using the absolute value of the negative discriminant. However, because of an error in the program, these roots are calculated using the negative quantity instead of the absolute quantity. An execution error results when an attempt is made to calculate the square root of the negative quantity. Tracing the flow will help locate the error.

```
100 REM THIS PROGRAM HAS AN ERROR
110 DATA 14, -2, 18.6, -12, 4, 36
120 READ A, B, C
130 D = B↑2 + 4*(A*C)
140 IF D <= 0 THEN 180
150 D = ABS(D)
160 PRINT 'ROOTS ARE' (-B+SQR(D))/ (2*A), (-B-SQR(D))/(2*A)
170 GO TO 120
180 PRINT 'ROOTS COMPLEX'
190 GO TO 160
200 END
```

EXERCISE 14 - 2:

1. Enter and RUN the QROOTS program.
2. When execution terminates because of an error, execute the program again by entering:

   TRACE FLOW, D

   The error can be corrected by changing the GO TO statement in line 190 to:

   190 GO TO 150

STEP Command

Another method of debugging is to execute the program statements step-by-step. If the STEP command, rather than the RUN command, is entered to start execution, the system pauses following the execution of each statement. As each pause occurs, the line number of the statement just executed is printed.

User: STEP
System: STEP MODE AFTER 0100

During this execution pause, the following commands may be used: DISPLAY, EDIT, WRITE, GO, SET, WIDTH, LIST, SUSPEND, READ, HELP, LISTCAT, CONDITION, SYMBOLS. These are the same commands that may be used duirng a pause resulting from execution of a PAUSE statement or from turning on the INQUIRY REQ switch.

To continue execution after the pause, press the PROG START if no other key was pressed following the pause. If a command was entered during the pause, then execution is continued by entering the GO command.

EXAMPLE:

GO
GO RUN
GO STEP

Note: If only GO is entered, execution continues in the manner specified by the command that started execution.

DISPLAY and SET Commands

The DISPLAY and SET commands are used during an execution pause. When the DISPLAY command is entered, the current values of all variables or specified variables in the program are printed.

DISPLAY ALL
DISPLAY R1, S, A$

To print the values of array variables, the array variable names must be specified.

```
DISPLAY A$(4), M(*), X(1,3)
```

The DISPLAY command may also be entered after program execution has been terminated because of a STOP or END statement, or a terminating execution error.

The SET command is used during an execution pause to change the value of a program variable.

```
SET X = 30
SET A$ = 'EQUAL TO'
SET B (1, 6) = .54E-10
```

When program execution is continued, the value given the variable in the SET command is used whenever the variable is referenced.

EXERCISE 14 - 3:

1. EDIT and LIST the saved program EOQ.
2. RUN
3. Execute again by entering the STEP command. During the pause that occurs after the first statement is executed, enter the following DISPLAY command:

   DISPLAY S, C
4. With the SET command, change the value of S to 5.00 and the value of C to .25.
5. Continue execution by entering

   GO RUN

DISABLE and ENABLE Commands

The DISABLE command causes specified lines in the Work File to be ignored during subsequent program execution.

```
DISABLE 25, 55-75
```

If lines 25 and 55-75 are program statement lines, they will not be executed when the program is later executed. If they are the line numbers of a keyboard-generated file, they will not be used when the file is referenced by a GET statement.

When a SAVE command is entered, the disabled status of the specified lines is also saved.

The ENABLE command cancels the disabled status of the specified statements of data file lines in the Work File. The specified lines will again be used in a program execution:

ENABLE 55-75

If no line numbers are specified, all disabled lines in the Work File are again used during execution:

ENABLE

EXERCISE 14 - 4:

1. EDIT the saved CUFUN program.
2. LIST
3. DISABLE 140-160, 190-230 (These are the statement lines related to limiting the output to 50 lines per page.)
4. LIST (Note the asterisk preceding each disabled line.)
5. RUN
6. ENABLE
7. RUN

Section 15 Program Modification Commands

Performance Objectives:

- Create a program by merging statement lines in the Work File with statement lines from a saved program
- Restructure a program by renumbering statement lines
- Relabel variables in a program
- Modify program lines with the CHANGE command

PROGRAM MODIFICATION COMMANDS

Program modification commands permit changes in the program that is in the Work File. These commands are:

MERGE
DELETE
CHANGE
EXTRACT
RENUMBER
RELABEL

In the following exercise the MERGE command is used to combine the program in the Work File with a segment of a program saved in the File Library Area. The program is then further modified and the RENUMBER command is used to renumber the lines.

EXERCISE 15 - 1:

1. EDIT and LIST the saved CUFUN program. Note the line numbers of the subroutine statements. In the CUFUN program shown in this text, these line numbers are 190-230.
2. EDIT and enter the following POWERS program:

   ```
   100 FOR N = 1 TO 100
   110 PRINT N, N↑2, N↑3
   120 NEXT N
   130 END
   ```

3. MERGE the subroutine from the CUFUN program with the POWERS program by entering the following command:

   MERGE CUFUN, 190-230 (use the line numbers of the subroutine statements in your CUFUN program.)

   The specified line numbers from the CUFUN program are entered into the Work File following the last line of the Work File.
4. LIST
5. Insert the following statements:

   ```
   112 L = L + 1
   113 IF L < 50 THEN 120
   114 GO SUB
   ```

6. Change the END statement to a STOP statement.
7. RENUMBER the program statements by entering the following command:

   RENUMBER

   The lines are renumbered in increments of 10.
8. RUN
9. SAVE

The subroutine can be deleted from the POWERS program with the DELETE command. The line numbers of the lines to be deleted must be specified with the command.

The subroutine can also be deleted by using the EXTRACT command. The EXTRACT command deletes all lines other than those lines specified with the command.

EXERCISE 15 - 2:

1. DELETE from the POWERS program in the Work File the subroutine and all statement lines that relate to the subroutine.

   DELETE 120-140, 170-210
2. LIST
3. EDIT the saved POWERS program.
4. EXTRACT those statement lines that are not part of the subroutine or related to the subroutine.

   EXAMPLE:

   EXTRACT 100, 110, 150

The name of the variable in the POWERS program can be changed with the RELABEL command. Both the current name and the new name of the variable must be specified in the command. These names are separated by a comma. The first variable name is the current name. The second variable name is the new name.

EXAMPLE:
RELABEL A, C

EXERCISE 15 - 3:

1. Change the name of the variable N in the POWERS program to P.
2. LIST

With the CHANGE command, lines in the Work File can be modified without having to rekey the line.

EXERCISE 15 - 4:

1. CHANGE line 100 in the Work File (POWERS program) to 100 FOR P = 25 TO 50 in the following way:
   a. Enter CHANGE 100 and press RETURN
   b. BKSP to the number 1
   c. Key 25 TO 50
   d. Press RETURN
   e. LIST
2. In line 110 CHANGE N↑2 to SQR(N) by entering the following:

   CHANGE 110 'N↑2', 'SQR(N)'
3. Press RETURN twice.

Section 16 Libraries

Performance Objectives:

Make library programs and data files available to other users

Restrict the availability of library programs and data files to other users

Prevent modification and listing of programs and data files made available to others

Display the names of programs and data files contained in a File Library Area

LIBRARIES

The One-Star Library

All of the programs and data files saved in your library are secure from use by anyone else. They may be used only when your password is in effect.

However, without endangering the privacy of your library, saved programs and data files may be made available to other system users. This is done with the POOL command, entered by keying POOL and a filename. A password and disk label may be included.

EXAMPLE:

```
POOL STOCK
POOL STOCK/JONES/DISK4
```

The filename specified must be a program or data file presently saved in your library. Your password and the disk-label must be specified only if your password is not in effect.

To use a program or data file that you have pooled, another system user need only reference the filename preceded by a single asterisk.

EXAMPLE:

```
EDIT * STOCK
LIST
RUN
```

He can EDIT, LIST, RUN, ALLOCATE, and even SAVE this program or data file in his own private library. He can modify the copy he has edited into the Work File, but he cannot modify or delete the program or data file as it exists in your library.

Because a pooled program or data file is referenced by a single asterisk, it is said to belong to the *one-star library*. The one-star library consists of all those programs and data files that various password holders have specified with the POOL command. Physically, programs and data files exist only in individual private libraries. However, if pooled, they may be used by any system user and are always referenced by the filename preceded by a single asterisk. The user who contributed the program or data file may still reference it by its filename without the preceding asterisk.

LISTCAT Command

As programs and data files accumulate in a private library, it becomes harder to remember what has been saved. For a listing of all filenames saved in your library, enter the LISTCAT command.

EXAMPLE: LISTCAT

The LISTCAT command followed by a single asterisk produces a list of all programs and data files contributed to the one-star library, regardless of the disk on which they are physically located.

EXAMPLE: LISTCAT *

EXERCISE 16 - 1:

1. LISTCAT
2. POOL the program named CUFUN.
3. LISTCAT *. (Note that CUFUN is listed in this library.)
4. Sign OFF. Your password will no longer be in effect.
5. Enter the following command:
   RUN *CUFUN
   The program will be executed even though no password is in effect. It is not necessary to EDIT a saved program before executing it.

PROTECT Command

Another system user may be prevented from saving or even listing a program (but not a data file) that you have contributed to the one-star library. This added protection is given to your one-star program with the PROTECT command.

EXAMPLE:

PROTECT *CUFUN

Once you have protected a one-star contribution, other system users are limited to the RUN, STEP, EDIT, TRACE, and ALLOCATE command when using this program. These restrictions, however, do not apply when your password is in effect.

If your password is not in effect when PROTECT is entered, it must be specified with the disk-label in the command.

EXAMPLE: PROTECT *CUFUN/SESAME

To cancel the restrictions placed on users of your programs, enter the PROTECT command with the word OFF.

EXAMPLE:

PROTECT *CUFUN, OFF
PROTECT *CUFUN/SESAME, OFF

Programs and data files in your private library also may be protected. The PROTECT command, used in this manner, prevents you from inadvertently destroying or modifying programs or data. A library program or data file that is protected cannot be destroyed by a DELETE command (discussed later in this section) or modified by saving a changing version of the file under the same name. Nor can a protected file be modified by output from another program.

To protect a program or a data file in a private library, the filename is entered with the keyword PROTECT:

PROTECT CUFUN

This protection may be cancelled by entering the command with the word OFF.

PROTECT CUFUN, OFF

EXERCISE 16 - 2:

1. LOGON
2. PROTECT the program *CUFUN.
3. Sign OFF.
4. EDIT and then try to LIST *CUFUN.
5. LOGON
6. PROTECT the program CUFUN.
7. EDIT and then LIST CUFUN.
8. Change line 120 to 120 FOR X = -10 TO 20
9. Try to SAVE this modified version. An error message is printed not only because this program is protected but also because it is pooled.
10. Cancel the protection status of CUFUN and *CUFUN. Two separate PROTECT commands must be entered.

PULL Command

To remove a program or data file from the one-star library, enter the PULL command along with the name of the program or data file to be removed. The filename is not preceded by an asterisk. You can PULL only those programs and data files that you contributed to the one-star library. If your password is not in effect, it must be specified with the disk-label.

EXAMPLE:

PULL CUFUN
PULL CUFUN/JONES

The PULL command must be entered anytime you want to modify a program or data file that you contributed to the one-star library. After modification, you can again POOL the program or data file.

The PULL command does not affect the specified program or data file as it exists in your own library.

EXERCISE 16 - 3:

1. PULL CUFUN
2. LISTCAT * (Note that CUFUN is no longer listed in the one-star library.)
3. RUN CUFUN. The program as it exists in your library has not been affected by the PULL command.

DELETE Command

If you no longer have use for a saved program or data file, it can be deleted from your private library with the DELETE command. The keyword DELETE is entered followed by the filename:

DELETE CUFUN

This command, however, will not be executed if the specified program or data file has been pooled in the one-star library or if it is protected.

You can also delete your entire library. When -ALL is specified, all files - program and data - are deleted from the library associated with your password and your password is deleted from the password directory. However, pooled or protected files are not deleted. Nor is the password deleted in this case.

EXERCISE 16 - 4:

1. DELETE CUFUN
2. LISTCAT

The Two-Star Library

The two-star library is a collection of programs and data files placed in the two-star library by any user of the system and available to any user.

A user, whether he has a password in effect or not, may save a program or keyboard-generated data file in the two-star library by entering a SAVE command followed by two asterisks and the filename.

SAVE **CUFUN

A program-generated data file is placed in the two-star library with the ALLOCATE command that is entered when the file is created.

ALLOCATE **CURRENT(DATA),NEW

A program or data file saved in the two-star library is referenced by the filename preceded by two asterisks.

EDIT **CURRENT
RUN **CUFUN

Unless the referenced two-star file has been protected, any user can LIST the file and SAVE it in his own library. He can modify the file as it exists in the two-star library and even DELETE it.

A two-star library file can be protected with the PROTECT command. This protection cannot be cancelled.

PROTECT **CUFUN

Users are then prevented from listing and saving the file in their own libraries. They are also prevented from modifying the file as it exists in the two-star library. However, they may still DELETE the file.

A file in the two-star library is deleted with the DELETE command followed by two asterisks and the filename.

DELETE **DUFUN

For a listing of all files in the two-star library, enter the LISTCAT command followed by two asterisks.

LISTCAT *

EXERCISE 16 - 5:

1. Sign OFF.
2. Enter the program KILO to convert miles to kilometers.
```
EDIT KILO
100 PRINT 'MILES'
110 INPUT M
120 K = M* .621
130 PRINT K 'KILOMETERS'
140 GO TO 100
150 END
```
3. SAVE **KILO
4. LISTCAT**
5. LOGON
6. EDIT**KILO
7. SAVE KILO in your own library.
8. Modify the copy of **KILO in the Work File by changing line 130 to PRINT K 'KILOMETERS PER' M 'MILES'
9. SAVE this modified version of **KILO. The asterisks and filename must be specified.
10. LIST the program.
11. PROTECT **KILO
12. EDIT **KILO and then try to LIST
13. DELETE **KILO

UTILITY COMMANDS

The S/3 BASIC utility commands are used for the updating and routine maintenance of the system. They provide the facility to:

- initialize disk cartridges
- define System Work Areas on disks
- define, expand, pack, and relocate File Library Areas
- copy and relocate the System Program Area
- copy and relocate the HELP Text File
- make copies of disks

Some of these utility commands were used during the system generation procedure. These are the INITIALIZE, COPY-DISK, ASSIGN-WORKAREA, ASSIGN-LIBRARY, and CONFIGURE commands.

INITIALIZE	Prepares a disk for use on the system. The disk is checked for defective tracks, and an identifying label is placed on the disk.
ASSIGN-WORKAREA	Allocates a System Work Area on R1 and F1. To allocate this area on disk drive 2, R2 or F2 must be specified with the command.
ASSIGN-LIBRARY	Allocates space for a File Library Area on a specified disk.
CONFIGURE	Defines or modifies the configuration record on F1, that is, the record of the components that are part of the computer system.
COPY-DISK	Copies the entire contents of one disk onto another disk.

A specified area of a disk may be copied using the COPY-SYSTEM, COPY-LIBRARY, or COPY-HELPTEXT commands.

COPY-SYSTEM	Copies the System Program Area onto another specified disk or to a different location on the same disk.
COPY-LIBRARY	Copies the File Library Area of a specified disk onto another specified disk or to a different location on the same disk.
COPY-HELPTEXT	Copies the HELP TEXT from one specified disk onto another specified disk or to a different location on the same disk.

Two other utility commands have functions related to the Volume Table of Contents (VTOC). These are the VTOC-DISPLAY and VTOC-DELETE commands.

Every disk has a Volume Table of Contents and contains such information as the disk label, the number of disk cylinders initialized, and the size and track location of the System Work Area and the File Library Area. The VTOC is automatically created by the system when the disk is initialized and automatically updated whenever changes are made on the disk.

When the VTOC-DISPLAY command is entered, the disk label, number of cylinders initialized, and the Volume Table of Contents for the specified disk is printed.

```
VTOC-DISPLAY R1
```

The VTOC-DELETE command deletes a specified System/3 BASIC FILENAME entry or all of the BASIC FILENAME entries. (Here FILENAME refers to an area on disk, e.g., File Library Area, Work Area. It does NOT refer to the filenames which are entered by the user.) Once an area is deleted from the VTOC, the area is no longer accessible to the system.

The command to delete all FILENAME entries on a disk is as follows:

```
VTOC-DELETE-ALL  R1, disk-label
```

When none of the information on the disk is accessible to the system, the disk can be initialized and reused with new data.

To delete only a particular area, for example, the Work Area, the command is:

```
VTOC-DELETE  R1, disk-label, WORKAREA
```

Space that was occupied by a deleted area is available for other uses.

The last two utility commands discussed here deal with the File Library Area. These are the EXPAND-LIBRARY and PACK utility commands.

The EXPAND-LIBRARY command increases or decreases the amount of space allocated for the File Library Area on a specified disk. It does not move the File Library Area but adds or deletes the specified space at the end of the existing File Library Area.

The PACK command consolidates all BASIC program and data files contained in the File Library Area into one consecutive area at the beginning of the File Library Area and places all unused space at the end of the area.

Additional information about utility commands can be found in *System/3 BASIC Reference Manual.*

INITIAL SYSTEM GENERATION

During system generation, there are a number of options for organizing disk storage. For instance, the System Program Area can be housed on either F1 or R1. Another option is whether or not to produce a second back-up copy of the System Programs.

The three-phase system generation procedure described below illustrates only one way of organizing disks. In this procedure, the programming system is copied from the PID disk cartridge to F1; a back-up system is produced; and a File Library Area is assigned on R1. For information about variations in the procedure, see *System/3 BASIC Operator's Guide,* "System Generation Procedures".

Since system Generation involves mounting and removing disks and inserting printer forms, a copy of *System/3 BASIC Operator's Guide,* in which these procedures are described, should be kept available.

PHASE I	The PID cartridge is mounted on R1, and the System Program Area, the File Library Area, and the HELP Text File are copied from the PID cartridge to the disk on F1.
PHASE II	A new disk cartridge is mounted on R1. The System Program Area, the File Library Area, and the HELP Text File now on F1 are copied to R1 to produce a second back-up system. This phase may be by-passed.
PHASE III	A disk cartridge for housing a File Library Area is mounted on R1. System Work Areas are assigned on both R1 and F1, and a File Library Area is assigned on R1.

PHASE I

The PID cartridge will be mounted on R1; the System Program Area, the File Library Area, and the HELP Text File will then be copied from the PID cartridge to the disk on F1.

Materials Required:

System/3 BASIC Operator's Guide
A new disk cartridge or a scratch disk cartridge
(a used disk with information no longer needed)
Printer paper
The PID cartridge

Steps to be Performed:

Insert printer paper
Power on
Mount PID cartridge
Initial Program Load
INITIALIZE F1 disk
CONFIGURE
Secondary INITIALIZE
COPY-DISK
Remove PID cartridge

Insert Printer Forms

1. Identify the printer model on the computer and locate the printer controls.
2. Insert printer forms, according to the directions in the *System/3 BASIC Operator's Guide.*

Power On

1. Make sure switches labeled DISK DRIVE 1 and DISK DRIVE 2 are OFF.
2. Move POWER switch to ON.

Mount PID Cartridge

1. On a sheet of paper, copy the disk label shown on the cover of the PID cartridge.
2. Press in the DRAWER RELEASE LEVER on Disk Drive 1 and open the drawer.
3. Mount the PID disk cartridge on Disk Drive 1, following the procedure in the *System/3 BASIC Operator's Guide.*
4. Wait about 60 seconds for the disk to reach operational speed.

Initial Program Load (IPL)

The Initial Program Load (IPL) procedure is required to start the system operations and must be performed everytime the system is turned on. An IPL at system generation, however, is a little different from the normal IPL.

1. Move the switch labeled DISK SELECT up to REMOVABLE. This switch indicates the disk on which the system programs are located. (Remember, the PID cartridge with all the system programs is mounted on R1.)
2. Move the Program LOAD switch to the On position. Because this switch is spring-loaded, it returns automatically to its former position. Turning on this switch loads the control program called the Supervisor from the PID cartridge into main storage. The system responds by printing:

   5703 XM1 COPYRIGHT IBM CORP. 1970
   ENTER DATE - MM/DD/YY

3. Key in the correct date: Month/day/year. Slashes must be entered.

   EXAMPLE: 01/06/71

   To correct a typing error, press the ERASE key and then rekey the date. When the date is entered correctly, press the RETURN key. The system should print the following error messages:

   ERROR 535 WRONG OR NO ⟨WORK AREA⟩ ON R1 and F1
   ERROR 545 F1 NOT INITIALIZED
   ERROR 547 MINIMUM CONFIGURATION RECORD ASSUMED
   READY

If the system responds with a question mark or an upward arrow, the date was not keyed in the proper format. Rekey the date.

The errors described in these messages are corrected in the following steps.

INITIALIZE F1 Disk

Disk initialization is performed on new disks and on disks containing data that is no longer needed. Initialization involves labeling the disk.

1. Decide on a label for disk F1. This label must be no longer than 6 characters (alphabetic, numeric or both). The name SYSRES to indicate system residence would be appropriate.
2. Key the INITIALIZE command in the following format:

   INITIALIZE F1, disk-label

   EXAMPLE: INITIALIZE F1, SYSRES

   To correct a typing error, press the ERASE key and rekey the information. When the line is keyed correctly, press the RETURN key. The system prints:

   DISK INITIALIZED AND MOUNTED
   READY

   Anytime the system responds with a question mark or an upward arrow, the information was not keyed in the proper format. Rekey the command.

CONFIGURE

During this operation, a record of the components available on the computer is written on Disk F1. The group of components that make up a computer is referred to as the *configuration.*

1. Determine if there are any optional devices or an optional storage size in your computer system.

Minimum	Optional
8K (8K processing unit)	12K, 16K
2D100 (2 disks, 100 cylinders)*	2D200 (2 disks, 200 cylinders), 3D, 4D
13MP (13″ serial printer)	22MP (22″ serial printer) 13 LP (13″ bi-directional printer) 22 LP (22″ bi-directional printer) CRT (Cathode Ray Tube Display Station) CARD (Data Recorder)
8 CK (8 command keys)	16 CK (16 command keys)
KB1 (English - U.S.)	KB2 (Austrian/German) KB3 (Belgian/French) KB4 (Danish) KB5 (Norwegian) KB6 (Finnish/Swedish) KB7 (Spanish) KB8 (Portuguese/Brazil) KB9 (United Kingdom)

* Data is stored in concentric tracks on both sides of a disk. A cylinder is composed of two tracks: a track on one side of a disk and the track that occupies the same position on the other side of the disk. A disk with a 100 cylinder capacity has 100 tracks on each side.

2. Key the CONFIGURE command, listing the optional components on your system. Optional components can be listed in any order. Except for the last item in the series, each item must be followed by a comma and/or one or more blanks. If there are no optional components, key only the command CONFIGURE. The system then assumes the minimum configuration and writes on F1 a record of the minimum system components.

   EXAMPLE: CONFIGURE 12K, 2D200, 22MP, CRT, 16CK
   or
   CONFIGURE

   The System prints:
   READY

3. For a print-out of the system configuration as written on F1, key the CONDITION command.

   EXAMPLE: CONDITION

   The system prints the configuration record and a description of the system condition.

Secondary INITIALIZE F1 Disk

Secondary initialization is performed on F1 only if the disk capacity is greater than 100 cylinders. Since a minimum system was assumed during the first (or primary) initialization, only 100 cylinders were initialized. If a disk capacity of 200 cylinders was specified in the CONFIGURE command, the additional 100 cylinders can now be initialized.

Key the INITIALIZE command in the following format:

INITIALIZE F1, disk-label, SECONDARY

The disk-label must be the same label used during primary initialization.

EXAMPLE: INITIALIZE F1, SYSRES, SECONDARY

COPY-DISK

In this step the System Program Area, the File Library Area with the sample program, and the HELP Text File are copied from the PID cartridge on R1 to the disk F1.

Note: If you don't want the File Library Area of the HELP Text File to be contained on F1, then see the following sections in the *System/3 BASIC Reference Manual:* "COPY-HELPTEXT Utility Command", "COPY-LIBRARY Utility Command", "COPY-SYSTEM Utility Command".

1. Locate the disk label for the PID cartridge.

2. Key the COPY-DISK command in the following format and press the RETURN key:

   COPY-DISK R1, disk-label on PID pack, F1, disk-label on F1

   EXAMPLE: COPY-DISK,PID710,F1, SYSRES

   The System prints:
   PRESS SYSTEM START TO COPY F1, disk-label TO R1, disk-label

3. On the Console, lift the SYSTEM START switch. Because this switch is spring-loaded, it returns automatically to a center position. When the system completes the copy procedure, it prints:

   CYLINDERS COPIED: XXXX (number of cylinders)
   ALL COPIES COMPLETED, RE-IPL

Note: DO NOT IPL until the PID cartridge on R1 is removed.

Remove PID Pack

Remove the PID pack following the instructions in the *System/3 BASIC Operator's Guide.*

Note: Since an IPL must be performed, the system is not in operation and the REMOVE command is not keyed.

PHASE II

A new disk cartridge will be mounted on R1. The System Program Area, the File Library Area, and the HELP Text File now on F1 will be copied to R1 to produce a back-up system. (If a second back-up system is not desired, go to PHASE III).

Steps to be Performed: Mount Disk Cartridge
Initial Program Load
INITIALIZE Disk
COPY-DISK
Remove Back-up Cartridge

Mount Disk Cartridge

On R1, mount a disk cartridge to contain the back-up system.

Initial Program Load

Because the system programs are now on F1, the IPL is performed from F1.

1. Move the DISK SELECT switch to FIXED.

2. Lift the PROGRAM LOAD switch. The system prints:

   5703-XM1 COPYRIGHT IBM CORP. 1970
   ENTER CONFIGURE COMMAND OR PRESS PROG START KEY

   (Note that this message differs from the one received during the IPL from the PID cartridge. The message received now is the message normally printed during an IPL.)

3. On the keyboard, press the PROGRAM START key. The system prints:

   ENTER DATE MM/DD/YY

4. Key the date and press the RETURN key. The system should print the following error messages:

   ERROR 535 WRONG OR NO < WORKAREA > ON R1 and F1
   ERROR 543 R1 NOT INITIALIZED
   READY

Note: These errors are corrected in the following steps.

INITIALIZE R1 Disk

To initialize the disk on R1, perform the following procedure:

1. Decide on a label for the disk. The label can have up to 6 characters (alphabetic, numeric, or both).

2. *If R1 is a new disk,* key the command in the format shown below:

   INITIALIZE R1, disk-label
   EXAMPLE: INITIALIZE R1, BAKUP1

   The system prints:

   DISK INITIALIZED AND MOUNTED
   READY

*Note: If R1 is a scratch disk (*a disk containing information no longer needed), see *System/3 BASIC Reference Manual,* "VTOC-DELETE Command" before initializing.

Copy-Disk

1. To copy the System Program Area, the File Library Area, and the HELP Text File from F1 to R1, key the COPY-DISK command in the following format and then press the RETURN key:

   COPY-DISK F1, disk-label, R1, disk-label
   EXAMPLE: COPY-DISK F1, SYSRES, R1, BAKUP1

   The system responds by printing:

   PRESS SYSTEM START TO COPY F1, disk-label TO R1, disk-label

2. On the Console, lift the SYSTEM START switch. When the system completes the copy procedure, it prints:

   CYLINDERS COPIED: XXXX (number of cylinders)
   ALL COPIES COMPLETED, RE-IPL

Note: DO NOT IPL until the back-up system on R1 is removed.

Remove Cartridge

Remove the disk on R1. Because an IPL is required, the system is not in operation. Do not key REMOVE.

PHASE III

A disk cartridge for housing a File Library Area will be mounted on R1. System Work Areas will then be assigned on both R1 and F1, and a File Library Area will be assigned on R1.

Steps to be performed: Mount disk cartridge
Initial Program Load
INITIALIZE disk
ASSIGN a System Work Area
ASSIGN a File Library Area

Mount Disk Cartridge

On R1, mount a disk cartridge to house a File Library Area.

Initial Program Load

The IPL procedure must be performed after a copy procedure is order to start the system again.

1. Lift the PROGRAM LOAD switch.
2. On the keyboard, press the PROGRAM START key.
3. Enter the date and press RETURN.

INITIALIZE Disk

To initialize the disk now on R1, perform the following steps:

1. Decide on a label for the disk. The label can have up to 6 characters (alphabetic, numeric, or both).
2. If R1 is a new disk, key the command in the format shown below:

   INITIALIZE R1, disk-label

   EXAMPLE: INITIALIZE R1, JONESCO

Note: If R1 is a scratch disk (a disk containing information no longer needed), see *System/3 BASIC Reference Manual,* "VTOC-DELETE Command" before initializing.

ASSIGN-WORKAREA

To create a work are on both R1 and F1, key:

ASSIGN- WORKAREA

When the RETURN key is pressed, the system prints:

FILE ALLOCATION COMPLETED
READY

ASSIGN-LIBRARY

To assign a library area on R1, enter the ASSIGN-LIBRARY command in the following format:

ASSIGN-LIBRARY R1, disk-label

EXAMPLE: LIBRARY R1, JONESCO

The disk label must be the same label used to initialize the disk. When the RETURN key is pressed, the system prints:

FILE ALLOCATION COMPLETED
READY

TURNING ON THE CRT

To use the optional CRT as well as the Printer, turn on the CRT power switch (lower right front of CRT) and press Command Key 10. Both devices will function as output units.

This completes system generation. The System/3 BASIC is now an operable system. To execute the sample program copied from the PID cartridge, perform the following procedures:

1. Key
   EDIT **REGCOR

   The System prints

   **REGCOR COPIED TO WORK FILE
   BASIC PROGRAM FILE
   XXX LINES, XXX DISK UNITS IN FILE, DATE LAST MODIFIED XX/XX/XX
   READY
2. Key the LIST Command:

   LIST

   The system prints the program statements.
3. Key the RUN Command:

   RUN

   The system executes the program.

S/3 Model 6
Guide to BASIC
READER'S COMMENT FORM

SR29-5001-0

Your comments and your answers to the following questions will help us design and administer self-study courses that better suit your needs. If your answer to a question requires further explanation, or if you have additional information you think would be helpful, please use the space provided. Comments and suggestions become the property of IBM.

	YES	NO
Did you find the material in this text easy to read and understand?		
Did you find the material in the text well organized for self-study?		
Did you find the material in this text too technical?		
Does the text contain about the right amount of information?		
Do you feel any particular topic should be added or emphasized?		
Do you feel any particular topic should not have been included?		
Did you perform the exercises in the text as directed?		
Did you write or modify programs as directed?		
Have you any previous experience in programming? If so, specify the programming language below.		
The objective of this course is to teach you to operate and write programs for the System/3 Model 6 using System/3 BASIC. Do you feel the course met its objective?		

COMMENTS

SR29-5001-0

YOUR COMMENTS PLEASE...

Your answers to the questions on the back of this form, together with your comments, will help us produce better educational materials for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

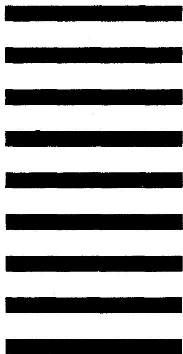
Cut Along Line

Fold | Fold

FIRST CLASS
PERMIT NO. 66167
CHICAGO, ILL.

BUSINESS REPLY MAIL

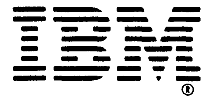
No Postage Stamp Necessary if Mailed in the United States



POSTAGE WILL BE PAID BY...

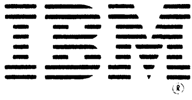
IBM Corporation
Education Development
301 East Erie Street
Chicago, Illinois 60611

Fold | Fold



International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York 10601
(USA only)

IBM World Trade Corporation
821 United Nations Plaza, New York, N.Y. 10017
(International)



International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York 10601
(USA only)

IBM World Trade Corporation
821 United Nations Plaza, New York, N.Y. 10017
(International)

SR29-5001-0

S/3 Model 6 Guide to BASIC Printed in U.S.A. SR29-5001-0