DataGeneral

basic BASIC

069-000003-00

basic BASIC

(An Introduction to BASIC)

069-000003-00

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software.

Ordering No. 069-000003

Printed in the United States of America
Revision 00, December 1978

NOTICE

basic BASIC
(An Introduction to BASIC)
069-000003

Revision History:

093-000088

Original Release - June 1973
First Revision - February 1976

069-000003

Original Release - December 1978

This revision changes only the part number; the text is identical to manual 093-000088 revision 01.

TO THE READER

Computers have been cloaked in mystery since their invention. This manual is aimed at clearing away some of the mystery and proving that you can use a computer. Remember, a computer is just an electronic machine which can perform complex calculations at high speeds. However, a computer can only do what it is told to do and you will be learning how to command one. Do not worry about mistakes, they cannot hurt the computer.

This manual introduces the BASIC computer language and covers its elementary commands and statements. When you become familiar with basic BASIC and would like to write more complex programs, please read Data General's Extended BASIC User's Manual.

We have designed this handbook for use at a computer terminal, so you can try the exercises which follow the concepts as we present them. Each section of the manual introduces a new idea, and ends with a simple exercise or two, which you should try before proceeding; you can thus progress at your own pace. (If you really get stuck, Appendix A includes one set of answers; use these only after you have tried the exercise.)

This handbook uses the following symbols:

Symbols	Meaning
*	Asterisk
)	Carriage Return
△	Space
( )	Parentheses
[ ]	Brackets

Handbook Symbol Table.

1. Parentheses and brackets are always interchangeable.

2. In the program examples, underlined words have been typed by a person. BASIC's responses are not underlined.

```
*LIST)
0010 REMARK - INTRO PROGRAM
0020 INPUT A
0030 PRINT A
0040 END

*RUN)
 ? 5432)
 5432

END AT 0040
*
```

You type LIST, a carriage return, then RUN, a carriage return, then 5432 and a carriage return. The computer prints everything else.

3. BASIC language keywords appear in CAPITAL letters.

4. Any words in **boldface** type in this handbook have meanings unique to the computer field. The index/glossary defines these words.

CONTENTS

CHAPTER 5 - DECISIONS & LOOPS

CHAPTER 6 - FOR/NEXT LOOPS

CHAPTER 7 - NUMERIC SUBSCRIPTING

CHAPTER 8 - STRINGS

APPENDIX A - PROBLEM ANSWERS

APPENDIX B - ERROR MESSAGES

APPENDIX C - PROGRAMMING ON MARK-SENSE CARDS

INDEX/GLOSSARY

Welcome to the fascinating world of computers. Do not be afraid to try out your ideas - experiment. This computer business should be fun!

CHAPTER 1
COMPUTER CONCEPTS

WHAT IS A COMPUTER?

A computer is a machine used to solve problems. The machine accepts information, processes it, and prints the answer. Three main units within a computer control these three tasks.

The three main units of a computer are the **Input/Output Unit**, the **Central Processing Unit** and the **Main Memory Unit**. Each unit performs specific functions, and information moves between them (Figure 1-1).

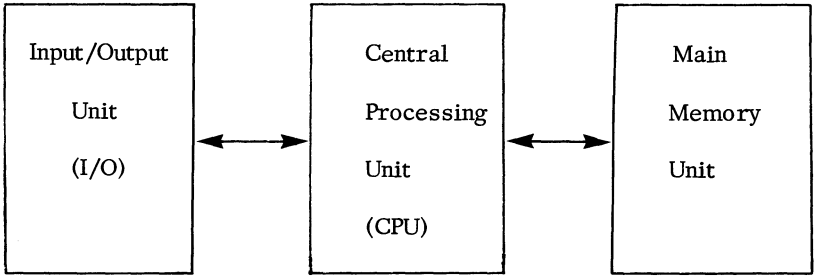


Figure 1-1. Information passes between the Three Main Units of a computer.

The computer's heart is the Central Processing Unit (CPU). The CPU deciphers your instructions to the computer, performs calculations, and directs the other units to complete your requests.

The Input/Output (I/O) unit controls the devices which feed (**input**) information to the computer and print (**output**) information back to you. Some **I/O devices** are teletypewriters, high-speed line printers, paper tape readers, and magnetic tape drives.

The Main Memory unit is a set of **addressable locations** which store instructions and information (called **data**) from the user. These locations resemble the mailboxes in a post office. Each mail box has a label and the mailman places letters in the proper

box. The CPU stores your information in a memory mailbox, keeps track of the mailbox label (called a **memory address**), and retrieves the data when necessary (Figure 1-2).

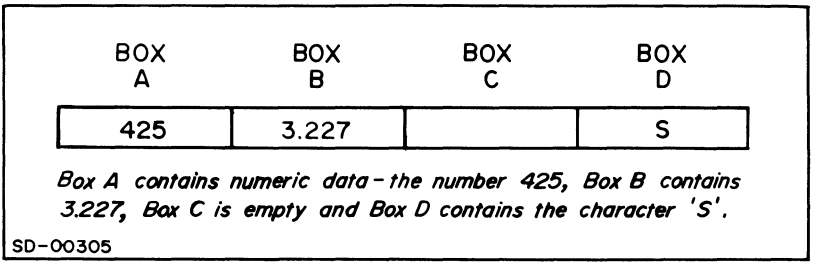


Figure 1-2. Data in a memory mailbox.

Through the I/O unit, the CPU obtains **data**, from the user. The CPU stores this information in the Main Memory unit. Whenever the CPU does calculations, it stores the results in the Main Memory unit.

WHAT IS BASIC?

BASIC is a computer language. You can converse with a computer in BASIC and instruct the computer to perform some task. BASIC stands for Beginner's All-purpose Symbolic Instruction Code and was first developed at Dartmouth College. BASIC uses familiar words and symbols so you can learn it quickly and easily.

To use BASIC, your computer must have a BASIC **language interpreter.** This interpreter is a set of instructions which interprets the familiar words and symbols of the BASIC language into commands which the computer executes. Data General supplied a BASIC language interpreter with your computer.

TALKING TO BASIC

You can type instructions in BASIC on a communications (I/O) device: either a **teletypewriter** (Figure 1-3), or a **cathode ray tube** (CRT) **display** (Figure 1-4).

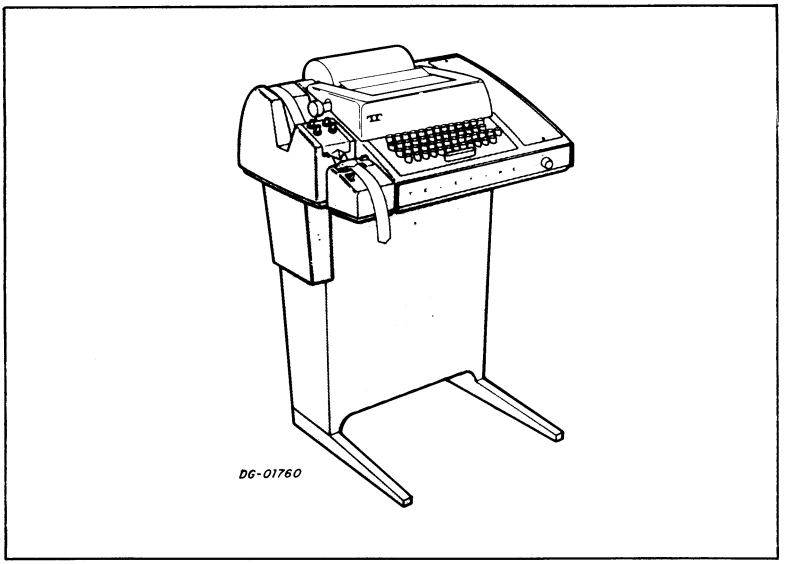


Figure 1-3. You can talk to a computer using a teletypewriter.

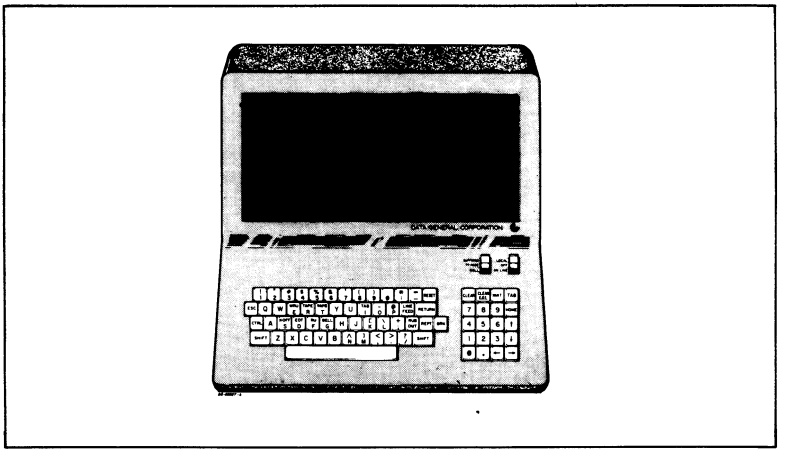


Figure 1-4. BASIC also understands instructions typed on a CRT display.

We will refer to your communications device as a **terminal.** It connects to the computer either directly or by telephone lines. Figure 1-5 contains a diagram of a terminal connected to a computer.

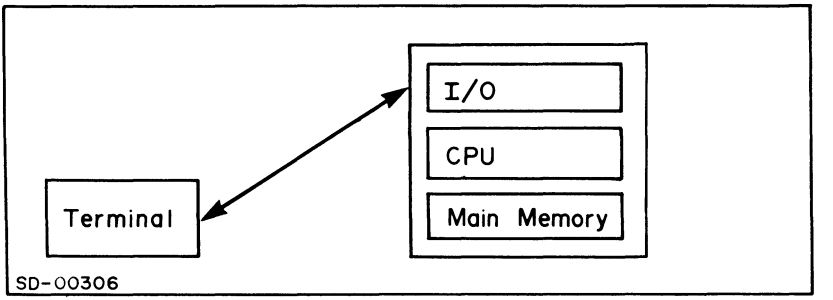


Figure 1-5. Your terminal connects to the I/O Unit of the Computer.

A terminal keyboard resembles a typewriter keyboard, but has several special keys. These special keys include:

the ESCape Key	interrupts the execution of the current calculation or task.
the RUBOUT Key	erases characters typed by mistake.
the SHIFT Key	works with other keys to create special characters which we will explain later.

Do not confuse alphabetic and numeric keys. Your terminal includes keys for the 10 numeric digits and you should use these keys for typing numbers, i.e., numeric 1 instead of lower case alpha L and numeric 0 instead of alpha O. Although these characters may appear similar on a printed sheet, BASIC interprets them differently.

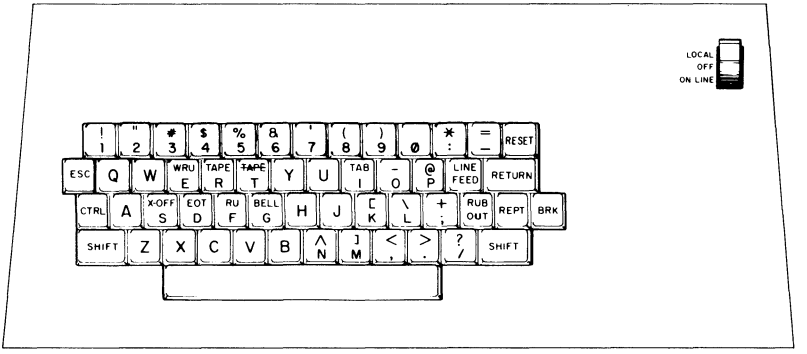


Figure 1-6. Special Keys on a DGC Model 6012 display keyboard.

Figure 1-6 illustrates a Model 6012 Keyboard. Other terminal keyboards are similar, though some of the special keys may be in different places.

You can also communicate with BASIC using **mark-sense cards** and a **card reader.** You mark these cards with a special code and stack them in the card reader, which reads them one at a time and sends their instructions to the computer. These cards can be used repeatedly; you may add or change cards to revise your program. Appendix C explains the code for these cards.

SHARING A COMPUTER

In a **time-sharing system,** many terminals share the same computer, and each terminal takes a turn using micro-seconds of computer time. The computer can process information so rapidly that each terminal user appears to have sole access to the system. You may have contact with only one terminal, but that terminal is part of a much larger system you never see (Figure 1-7).

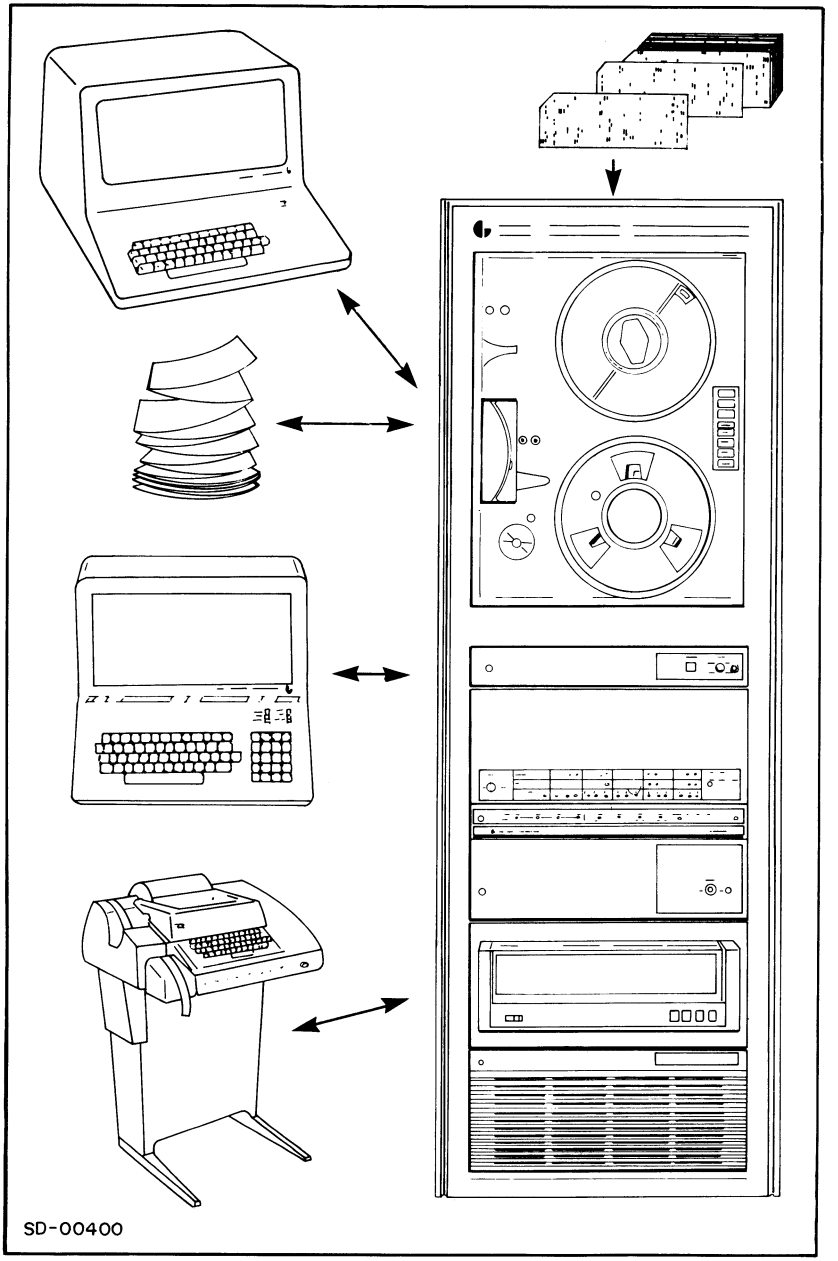


Figure 1-7. Many users share this NOVA II.

END OF CHAPTER

CHAPTER 2

AT THE TERMINAL

A computer will not do anything unless you ask it to. It will sit idle and be perfectly happy for minutes, or hours, or years. Therefore, you must learn the **commands** which you can type to tell BASIC what to do.

But, before we start listing commands, let's get you onto a terminal.

LOGGING ON

To start using BASIC, turn the terminal switch to LINE, then press the ESCape key on the keyboard. BASIC will print a **log on** message. If your message includes an asterisk prompt (*), you are logged on and may start typing instructions. Whenever you want a program to stop running, for whatever reason, hit the ESCape key. BASIC will reserve your program without loss, and print the asterisk prompt.

Some systems need account identification information to prevent unauthorized people from using the computer. If so, when you press the ESCape key, BASIC will print:

ACCOUNT ID:

and wait until you type a valid ID and a carriage return.

Suppose your **system manager** assigns you an account ID of HASK. Directly after BASIC prints "ACCOUNT ID:" type "HASK" and carriage return. Your account ID will not appear on your paper (or CRT screen) for security reasons.

If you type the wrong account ID, BASIC will indicate an unrecognizable ID and ask for a correct one. After you have typed a correct account ID the system will print log on, or **sign on,** information, then an asterisk indicating that it is ready for instructions. See Figure 2-1 for a sample log on procedure.

```
DGC READY
ESCape Key)                              (esc))
ACCOUNT-ID: HASK)                        (id not echoed)
     08/26/75 15:43 SIGN ON, 04

*
```

Figure 2-1. Hello, BASIC.

LOGGING OFF

When you wish to stop using the system or **log off**, type the command:

```
BYE)
```

On some systems after you type the command BYE, BASIC will print log off, or **sign off**, information which looks something like Figure 2-2.

```
*BYE)
     08/26/75 15:43 SIGN OFF, 04
     08/26/75 15:43 CPU USED, 0
     08/26/75 15:43 I/O USED, 0

DGC READY
```

Figure 2-2. Goodbye, BASIC.

BASIC will then sit idle until you or someone else presses the ESC key on the terminal.

Exercise 2-1. See if you can log onto your system, then log off.

THE PRINT COMMAND

Since BASIC is an interactive language, you and the computer carry on a dialog. The first BASIC command is PRINT, which means the same in BASIC as it does in English. After an asterisk prompt, which means it's your turn to talk, type PRINT, a number to be printed, and a carriage return. BASIC will print the number:

```
*PRINT 7)     7
*
```

(Remember, underlined sections of listings are those you type and the symbol) indicates the carriage return key which always ends a command.)

To tell BASIC to print letters and spaces rather than numbers, enclose these characters in quotation marks. BASIC will print exactly what you type within the quotes:

```
*PRINT "I AM A COMPUTER")  I AM A COMPUTER

*PRINT "I CAN'T SPEL")  I CAN'T SPEL

*
```

You may include both numbers and characters inside quotation marks:

```
*PRINT "REVOLUTION - 1776")  REVOLUTION - 1776

*
```

Do you see the difference? Numbers do not have to be in quotes but if you want BASIC to PRINT letters, punctuation or other characters, you must enclose the whole **string** in quotation marks.

BASIC carries out, or **executes** the PRINT command immediately. Notice that the carriage return signals BASIC to print the result. The command is not stored for re-use. Later, you will learn how the PRINT keyword can be stored in a BASIC program.

Exercise 2-2. Try PRINTing some values and messages.

Printing Numbers

BASIC converts very large or very small numbers to **exponential form.** Let's look at some numbers and then we will explain the conversion.

```
*PRINT 2000000000000000)   2E+15

*PRINT 1256E+7)   1.256E+10

*PRINT 145.745)   145.745

*PRINT .00000000432)   4.32E-09

*PRINT -324.56743)   -324.567

*PRINT -17.43985762527)   -17.4398

*PRINT 50000000000112)   5E+13

*
```

The letter E means "times 10 to the power of". BASIC converts all numbers of more than 6 digits to exponential form. BASIC accepts numbers from approximately 5.4×10^{-79} to 7.2×10^{75}. Notice that the letter E may be used in a number but a comma, to separate thousands, may not be used. Table 2-1 illustrates the relationship between standard notation, scientific notation and exponential form.

Table 2-1. Number Relationships

Standard	Scientific	Exponential
1,000,000	1×10^{6}	1E+06
10,000,000	1×10^{7}	1E+07
100,000,000	1×10^{8}	1E+08
.0000001	1×10^{-7}	1E-07
.00000000001	1×10^{-11}	1E-11

Exercise 2-3. Experiment with exponential notation on your terminal.

Using PRINT as a Calculator

You can use BASIC as a calculator with the PRINT command.

```
*PRINT 2+3)     5
*PRINT 4-1)     3
*PRINT 5-2+10-17)   -4
*
```

Exercise 2-4. Locate the plus (+) and minus (-) keys on your terminal and try some addition and subtraction.

While the arithmetic cross (x) means multiplication, BASIC would get confused if x were used for both an alphabetic character and a multiplication command. Therefore, BASIC uses an asterisk (*) for multiplication, and reserves x for alphabetic use. Most terminals do not contain a divide key so a slash (/) is reserved for division.

```
*PRINT 7*8)     56
*PRINT 9/3)     3
*PRINT 8*2/4)     4
*
```

Exercise 2-5. Try some multiplication and division at your terminal.

You can combine multiplication, division, subtraction and addition in the same PRINT command, and use parentheses as in algebraic statements. BASIC and arithmetic evaluate terms in the following order:

1. Any expression within parentheses - if nested, i.e. (3+(4*6)), the innermost parentheses are evaluated first.

2. Multiplication and division - from left to right, equal priority.

3. Addition and subtraction - from left to right, equal priority.

```
*PRINT (2+3)/5-7*2)  -13
*PRINT 4/7+8*2-4.35)  12.2214
*
```

BASIC evaluates the first example in this order:

1. parentheses: (2+3)
2. leftmost division: 5/5
3. rightmost multiplication: 7*2
4. subtraction: 1-14

Exercise 2-6. Try some examples combining addition, subtraction, multiplication and division.

You can print both messages and the results of calculations with a single PRINT command. Separate the items in the PRINT command with either commas or semicolons.

```
*PRINT "2 PLUS 3 IS";2+3)  2 PLUS 3 IS 5
*PRINT 4*5, "IS THE AREA")  20          IS THE AREA
*
```

Exercise 2-7. Try some problems combining calculations and messages. Experiment with commas and semicolons to see how they affect the format of your output.

END OF CHAPTER

CHAPTER 3

PROGRAMMING

PROGRAMS

A **program** is an ordered set of instructions, called **statements**, which define tasks for a computer. You will learn to create programs with BASIC statements.

Each BASIC statement begins with a **line number**. When the computer executes a BASIC program it performs the statement with the lowest line number first and proceeds to the next higher number. While any integer between 1 and 9999 may be used for a line number, programs in this manual will generally count line numbers by 10s.

Figure 3-1 contains a PRINT statement which looks very similar to a PRINT command. We will look more closely at the PRINT statement later.

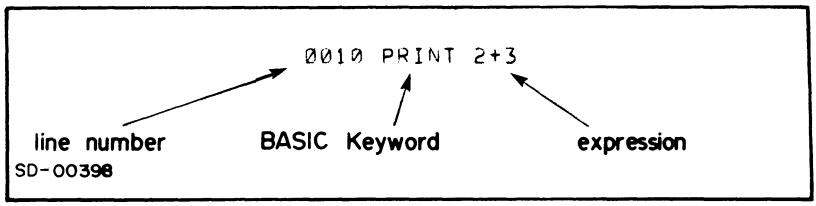


Figure 3-1. The PRINT statement contains a line number.

FLOW CHARTS

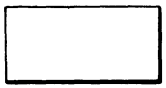
Before a programmer writes a program, he will often draw a symbolized diagram of its steps. This diagram, called a **flow chart**, maps the logic steps in a program. With simple programs, you may not need a flow chart; you will find more complex programs easier to write if you flow chart them first. In this handbook, we have provided a flowchart to clarify each BASIC program.

Certain geometrical shapes have evolved which represent specific computer actions. We have described some of these shapes below, and will explain others when you need them.

START OR STOP BOX

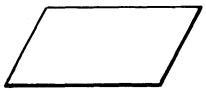
An elongated oval shows the beginning or ending point in a program.

PROCEDURE BOX



A rectangular box indicates an event or procedure. Arithmetic calculations are shown in a procedure box.

INPUT/OUTPUT BOX

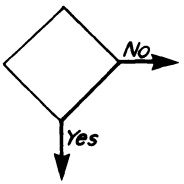


The parallelogram indicates input typed from your terminal and output the computer will print back to you.

ARROW

The arrow shows the direction logic will move through your program. Direction within a flow chart usually moves from top to bottom and from left to right.

DECISION BOX



A decision box contains a question which has a "yes" or "no" answer, such as "is A > 0?" If the answer is "yes" the computer will follow the "yes" arrow; if "no" the computer will follow the "no" arrow.

A good flow chart will keep track of the logic within your program and locate decision points and tests. If the logic has been planned, your program will be easier to write; your only worry will be translating a flowchart into BASIC. Also, if your program does not work, you can often find an error more easily in a flow chart than in the program itself. Just remember to keep your flow charts simple; put in the major steps and leave the details for the program.

WORKING STORAGE

When you sign onto BASIC, the system assigns you an area of memory called **working storage.** You use this part of memory like a giant chalkboard, to write, read and erase programs and data.

BASIC looks at every line you type as soon as you hit the carriage return. If you have entered a statement (a line number, BASIC keyword and any expressions or **arguments**), BASIC will store it in working storage. If you have entered a command (BASIC keyword, no line number) BASIC will execute it immediately without storing it. If you type something which is neither a valid statement or command, BASIC will ignore that line and print an **error message.**

Therefore, you can store all your BASIC statements in working storage. BASIC will leave them there until you explicitly erase or change them. When you have entered a complete set of statements, you have a program. You can type the command RUN (explained below), and BASIC will perform the program steps in sequence.

SOME COMMANDS FOR PROGRAMMING

Some commands which you will need for BASIC programs are:

LIST

When you have completely typed or **coded** a program, the LIST command will print the contents of working storage - all the program statements. This printing of a program is called a **listing.**

RUN

To execute your program, type the command RUN. The computer will RUN your BASIC statements in order, from the lowest line number.

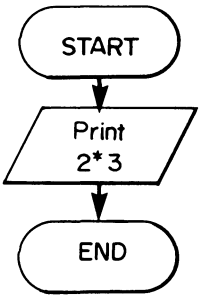
NEW

After your program has been run and you want to work on another, the NEW command will erase your working storage blackboard. It is good practice to type the NEW command before starting a program.

But enough explanation - this information will make more sense as you use it. Let's start programming!

OUR FIRST PROGRAM

We will start with a very simple program, in Figure 3-2, which multiplies 2 by 3. This program demonstrates PRINT in a BASIC statement instead of as a command.




```
*10 REM - PRINT 2 TIMES 3)
*20 PRINT 2*3 )
*30 END)
*LIST)
0010 REM - PRINT 2 TIMES 3
0020 PRINT 2*3
0030 END

*RUN)
 6

END AT 0030
*
```

Type in the program and tell the computer to list it.

The computer prints a listing.

You type the command to execute the program. The computer prints the result and an END message.

SD-00262

Figure 3-2. A long way to print "6".

Now let's look at the statements in our program.

Line 10 is a REMarks line. The BASIC statement REM stands for REMarks; with it you can write notes to yourself within a program. BASIC ignores all comments in a REM statement but prints them on every listing.

Line 20 uses the PRINT keyword in a statement. The line number makes a PRINT statement instead of a PRINT command; BASIC won't execute statements until you type RUN. A single program may include many PRINT statements.

Line 30 is an END statement which tells the computer that your program is finished and to return control to your terminal.

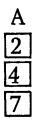
Exercise 3-1. Demonstrate that more than one PRINT statement can be used in a program. Write a program to divide 7 by 3 and multiply 6 times 4 and PRINT the results. If you make a typing mistake re-type the entire line including the line number.

VARIABLES

In BASIC, as in algebra, letters can represent numeric values. For a rectangle, you might choose W and L to stand for width and length in both BASIC and algebra; you would call W and L **numeric variables** in BASIC. Any letter which represents a number is a numeric variable.

In BASIC, the LET statement assigns values to variables. You can think of the variable name as a memory location, which holds the value of your variable. The LET statement can also change the value within this memory mailbox.

```
0020 LET A=2
0030 LET A=4
0040 LET A=2+5
```



Line 20 assigns the value 2 to memory address A.
Line 30 changes the value in memory address A to 4.
Line 40 changes the value in memory address A to 7.
The value in A changes with each LET statement.

You can name a numeric variable either a letter (A or F) or a letter followed by a digit (Q2 or R9). An arithmetic expression (J + Z) is not a valid variable name.

```
*10 LET A=7)
*20 LET A=9 )
*30 LET A=A+1)
*40 LET C9=A*4/3)
*50 LET C/2 =C9)
ERROR 02 - SYNTAX
*
```

Examine the program in Figure 3-3 which assigns values to A and B, adds the values, moves the sum to memory address C, and prints A, B, and C.

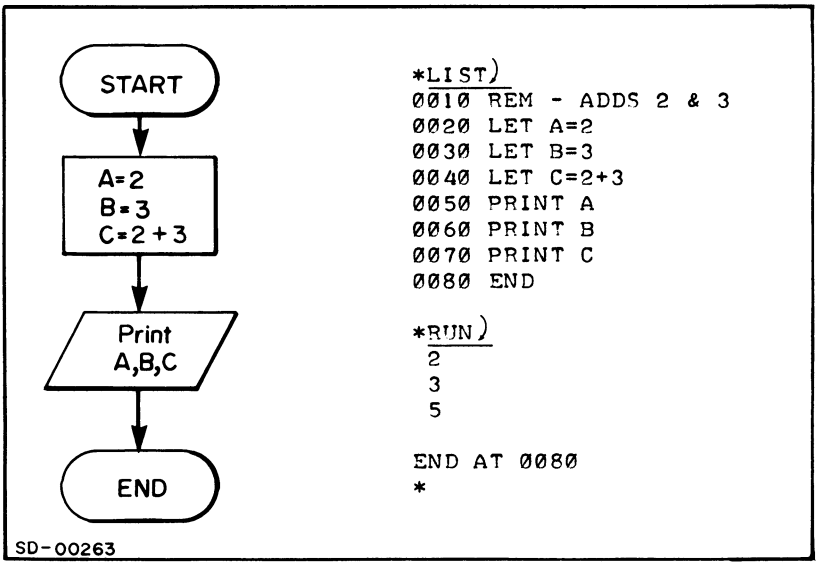


Figure 3-3. A Program Using Variables.

Exercise 3-2. Is 40 LET A + B = C a valid BASIC statement? Why or why not?

Exercise 3-3. Write and run a program which uses variables to print D if A = 2, B = 3, C = 4, and D = A*B-C.

EDITING

Everyone makes typing mistakes, so here's how to correct them. The easiest way to delete a character is with the RUBOUT key. The RUBOUT key prints a back arrow (←) on your terminal and erases the last character you have typed. You can continue pressing RUBOUT, and BASIC will erase characters right to left, one by one.

To erase the line you are typing use the SHIFT/L combination (hold down the SHIFT key and press L). BASIC will print a **backslash** (\ ) on your terminal and execute a carriage return. You can now retype the line.

Figure 3-4 contains examples of editing with both the RUBOUT key and the backslash.

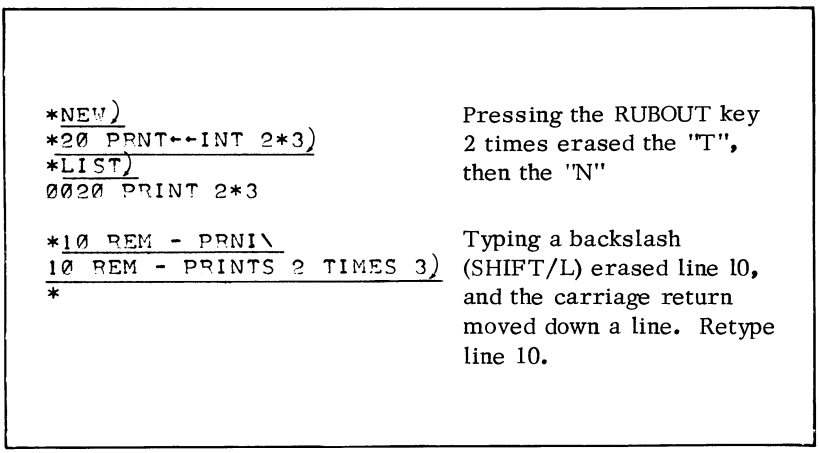


Figure 3-4. Editing Mistakes.

After you write a program, you may need to **debug** it (remove the programming errors). You can delete, add or change a program statement by its line number.

To erase a line within a program, just type its line number. To change a statement, retype its line number and the new statement. To add a statement, give it a line number between two existing line numbers. It will be easy to add line numbers if you have stepped them by 10s, as we have done in the examples. Programmers step their line numbers for just this reason - to add statements as they debug their programs (Figure 3-5.)

```
*LIST)
0010 REM - ADDS 2 & 3
0020 LET A=2
0030 LET B=3
0040 LET C=A+B
0050 PRINT C
0060 END

*20)                              Erase line 20
*35 LET D=2)                      Add new line between 30 and 40
*40 LET C=B+D)                    Change line 40
*LIST)
0010 REM - ADDS 2 & 3
0030 LET B=3
0035 LET D=2
0040 LET C=B+D
0050 PRINT C
0060 END

*
```

Figure 3-5. Editing What You've Got.

Another command to help you edit is RENUMBER. RENUMBER assigns line number 10 to the first statement in your program and renumbers the remaining lines in increments of 10.

The program in Figure 3-6 renumbers the program in Figure 3-5.

```
*LIST)
0010 REM - ADDS 2 & 3
0030 LET B=3
0035 LET D=2
0040 LET C=B+D
0050 PRINT C
0060 END

*RENUMBER)
*LIST)
0010 REM - ADDS 2 & 3
0020 LET B=3
0030 LET D=2
0040 LET C=B+D
0050 PRINT C
0060 END
*
```

Figure 3-6. Remember to RENUMBER.

The RENUMBER command is very handy after you have added or deleted statements in your programs.

Exercise 3-4. Type in the renumbered program in Figure 3-6 and edit it to let A = 5, B = 3, C = A, and D = A + B. Have it print A, B, C and D. Renumber, list and run your revised program.

FUNCTIONS

The BASIC language includes many different functions. Each function tells BASIC to perform an operation which would otherwise take several statements in your program. We generally use functions as expressions in BASIC statements; properly applied, they will save you many steps.

We will introduce three functions now. Other functions are explained later in this handbook.

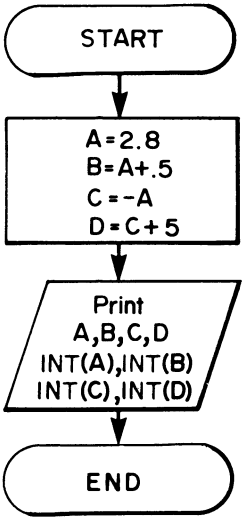
Integer Function

The INTeger function, INT(X), generates, or **returns**, a value equal to the greatest integer not larger than X.

If X is 3	INT(X) is 3
If X is 1.7	INT(X) is 1
If X is -2.2	INT(X) is -3

You can use INT(X) to see whether or not a number is an integer. If your number is an integer, INT(X) will equal X. To round numbers, add .5 and take the INTeger of the sum.

The INTeger function is used in programs where only integer data makes sense. In a population study for example, you would take the integer value of the number of people living in a certain area rather than publish the fact that 17.043279 people live within walking distance of bus terminals. Examine Figure 3-7 for a demonstration of the integer function.




```
*LIST)
0010 REM***DEMO OF INT FUNCTION***
0020 LET A=2.8
0030 LET B=A+.5
0040 LET C=-A
0050 LET D=C+.5
0060 PRINT A,B,C,D
0070 PRINT INT(A),INT(B),INT(C),INT(D)
0080 END

*RUN)
 2.8            3.3            -2.8           -2.3
 2              3              -3             -3

END AT 0080
*
```

SD-00264

Figure 3-7. The Integer Function.

Sign Function

The Sign function SGN(X), will tell you whether a number is positive, negative, or neither (0).

If X is positive	SGN(X) = +1
If X is zero	SGN(X) = 0
If X is negative	SGN(X) = -1

The program in Figure 3-8 prints the value, then the sign of A, B and C.

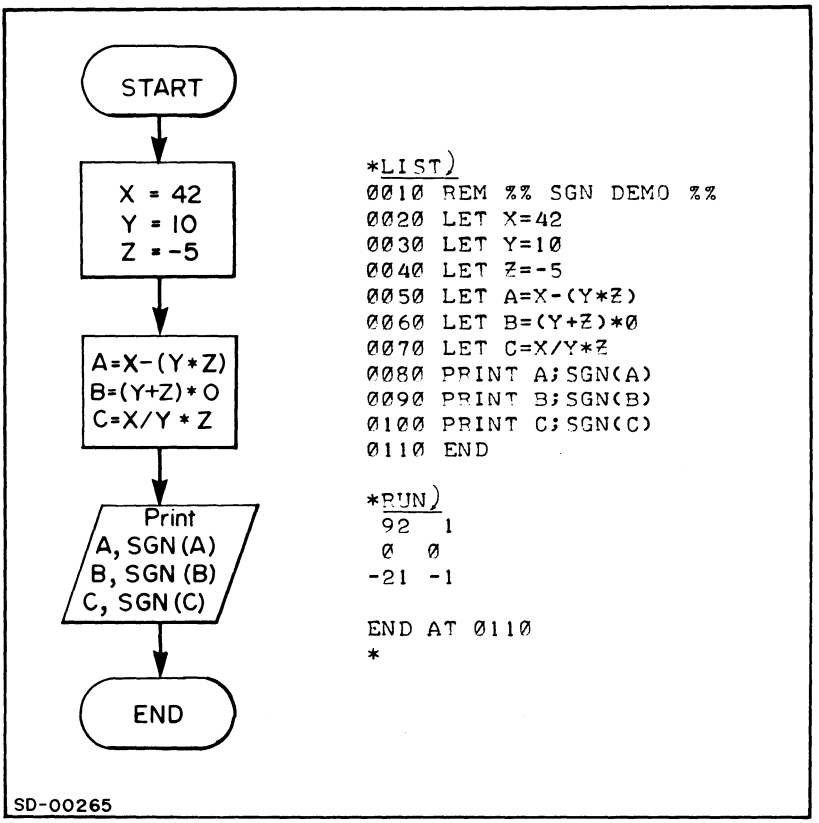


Figure 3-8. Finding the Sign.

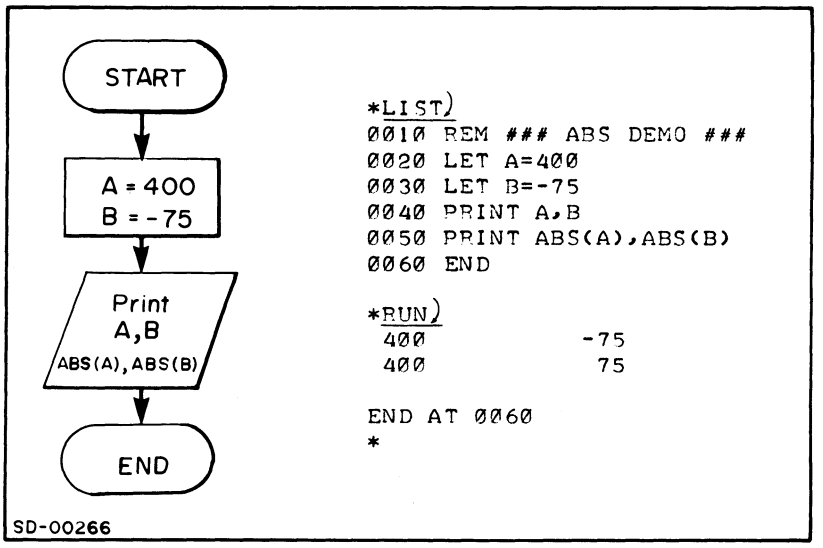
Use the Sign function when you don't care about the value of a number but want to know whether it is positive or negative. A credit checking program might take the Sign of someone's monthly totals and print "CREDIT" or "DEBIT" depending on the result.

Absolute Value Function

You might sometimes want the value of a number without regard to its sign. The absolute value function, ABS(X), returns the positive value of X.

```
ABS(1)  = 1
ABS(-1) = 1
```

Figure 3-9 contains a program using the absolute value function.




```
*LIST)
0010 REM ### ABS DEMO ###
0020 LET A=400
0030 LET B=-75
0040 PRINT A,B
0050 PRINT ABS(A),ABS(B)
0060 END

*RUN)
 400            -75
 400             75

END AT 0060
*
```

Figure 3-9. The Absolute Value Function.

You use the absolute value function when you want the difference between two numbers regardless of their sign. Sometimes road races are won on a target time basis, where the winner may be a few seconds early or late, but is closest to the target time. You calculate finalists by taking the absolute value of the finishing time minus the target time.

Exercise 3-5. Figure 3-10 is a flowchart for a program that will return a value of 1. See if you can write the program. Are all three functions necessary in this program? Do they have to be written in this order? There are some values for A which will not print 1. Can you tell what they are? Can you alter the program so A is always negative 1?

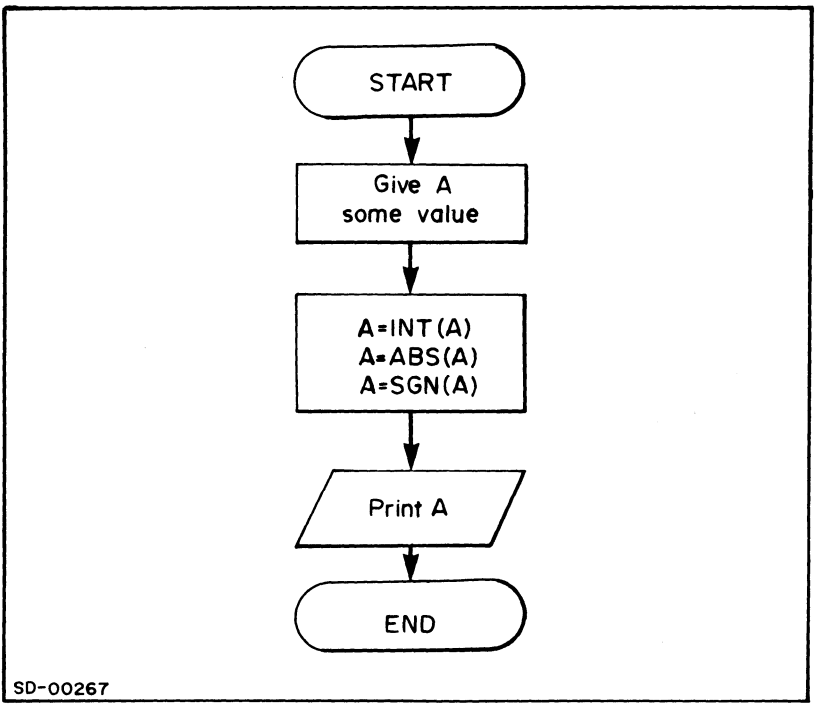


Figure 3-10. Flowchart for Exercise 3-5.

END OF CHAPTER

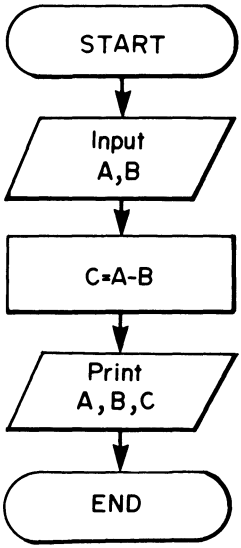
CHAPTER 4
WORKING WITH DATA

WHAT'S DATA?

Some BASIC programs will need specific information from you to run. This information is data. If a program calculates salaries, the hours worked and wage per hour would be the data; the program requires these numbers to run.

The LET statement assigns values to variables before you run a program. The INPUT statement allows you to assign values while the program is running. To use INPUT, type the keyword INPUT after a line number, and the variables for which you will supply values. Always separate INPUT variables with commas.

When BASIC encounters an INPUT statement while executing your program, it prints a question mark and waits for you to supply a data value. Therefore, with INPUT you can run the same program repeatedly using different data for each run (Figure 4-1).




```
*LIST)
0010 REM - INPUT COMMAND
0020 INPUT A,B
0030 LET C=A-B
0040 PRINT A,B,C
0050 END

*RUN)
 ? 8)    ? 3)
 8               3               5

END AT 0050
*RUN
 ? 5,2)
 5               2               3

END AT 0050
*
```

SD-00268

Figure 4-1. INPUTing Values.

When you enter data, after an INPUT question mark, you may separate values by either a carriage return or a comma. If you type improper data (alphabetic when BASIC is looking for numeric) BASIC will print a backslash-question mark (\ ?) and wait for valid information. If you type more data than your program needs, BASIC will print an error message.

Exercise 4-1. Write and run a program that will calculate and print the area of a rectangle if values for length and width are input by the user. Use the formula: area = length * width.

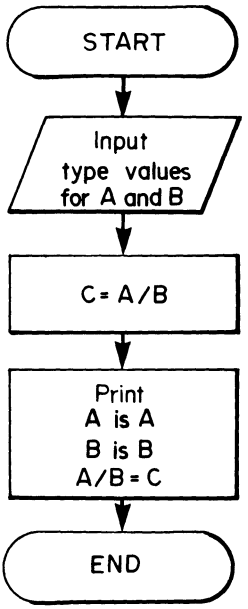
PROMPTING MESSAGES

Sometimes a question mark from an INPUT statement is not enough information. If you are not familiar with the program that is running and BASIC prints a question mark, you may not know what data the program needs. To clarify your own programs, you can insert a prompting message in quotation marks after an INPUT statement. BASIC will print this message instead of the question mark. After the closing quotes of the prompt, type a comma, then the variable, or variables, for which you want data.

You may want to clarify output as well. To do this, type your message in quotes after a PRINT statement. Type commas or semicolons between this PRINT message and each of your program's variables. (You'll find more on PRINT punctuation at the end of this chapter.) The program in Figure 4-2 uses prompts with both INPUT and PRINT statements.

Exercise 4-2. Write and run a program that will figure simple interest, and add it to principal. Your prompting messages should ask for principal, interest rate and number of years, then describe the interest accumulated and the new principal. Use the formula:

interest = principal * interest rate * number of years




```
*LIST)
0010 REM - INPUT WITH PROMPTS
0020 INPUT "TYPE THE VALUE OF A: ",A
0030 INPUT "TYPE THE VALUE OF B: ",B
0040 LET C=A/B
0050 PRINT "A IS ";A
0060 PRINT "B IS ";B
0070 PRINT "A/B= ",C
0080 END

*RUN)
TYPE THE VALUE OF A: 99)
TYPE THE VALUE OF B: 17)
A IS  99
B IS  17
A/B=            5.82353

END AT 0080
*
```

SD-00269

Figure 4-2. INPUT With Prompts.

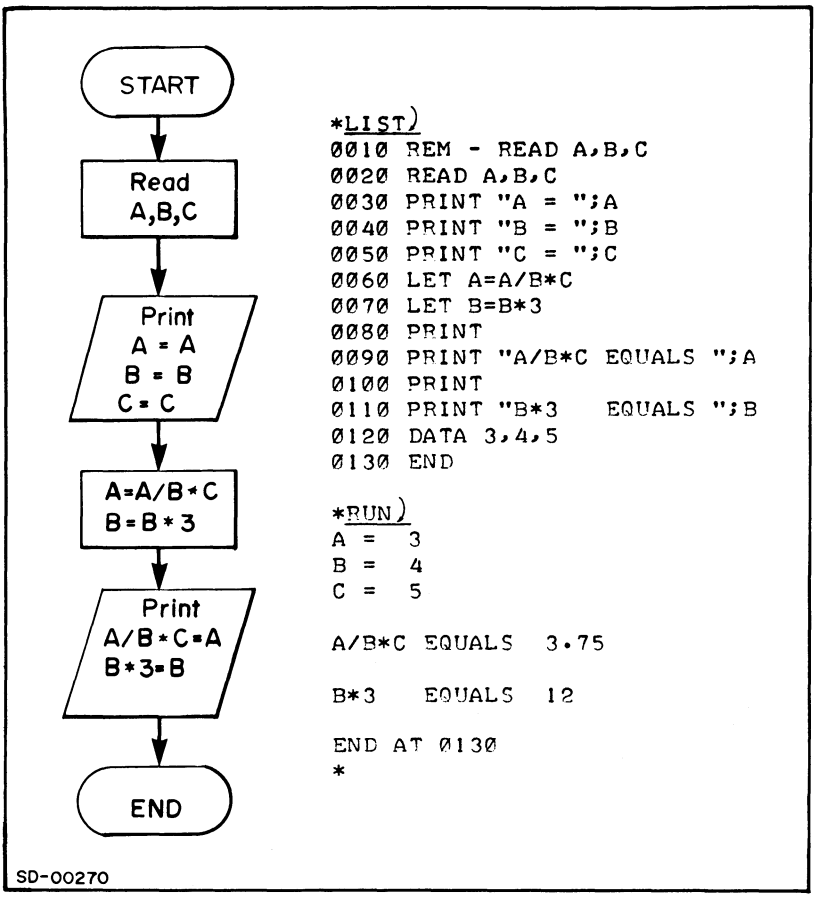
IF YOU KNOW YOUR DATA

With INPUT, you must interact with your program. The READ and DATA statements tell your program to read its own data, and run without asking questions. Use READ and DATA when you know the data before you run the program.

You can list variables in the READ statement, and list values for these variables in the DATA statement. The order of variables in READ is the order of retrieval from DATA. If the number of variables in the READ list exceeds the number of values in the DATA list BASIC returns an error message. Examine Figure 4-3 for a demonstration of the READ and DATA statements.

Neither READ nor DATA will work alone in a program; you must use both. The DATA statement often ends a program, although you can put it anywhere. BASIC keeps track of the DATA statements and always knows which data value is the next to be read.

Exercise 4-3. Write a program to figure salary using three data values - one for the employee number, one for hours worked, and one for wage per hour. Figure total salary by multiplying hours by wage. Print all four variables using prompts to make the output understandable. Change the values in the DATA statement and run the program again.




```
*LIST)
0010 REM - READ A,B,C
0020 READ A,B,C
0030 PRINT "A = ";A
0040 PRINT "B = ";B
0050 PRINT "C = ";C
0060 LET A=A/B*C
0070 LET B=B*3
0080 PRINT
0090 PRINT "A/B*C EQUALS ";A
0100 PRINT
0110 PRINT "B*3    EQUALS ";B
0120 DATA 3,4,5
0130 END

*RUN)
A =  3
B =  4
C =  5

A/B*C EQUALS  3.75

B*3   EQUALS  12

END AT 0130
*
```

Figure 4-3. READ and DATA Statements.

REMEMBER TRIG?

If you have had trigonometry, you can use BASIC's trigonometric functions. Skip this section if you are not familiar with trigonometry.

BASIC trig functions use radians ($180^\circ = \pi$ radians) to measure angles. The four BASIC functions to calculate trigonometric relationships of angles are:

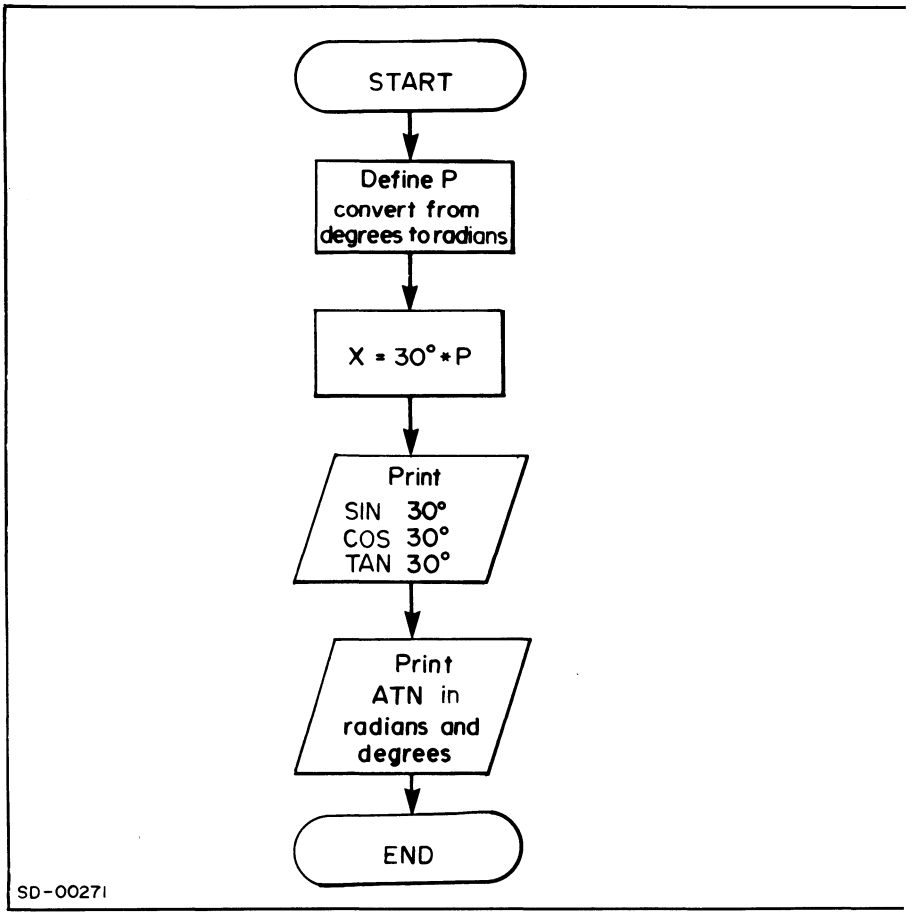


Figure 4-4. Trigonometry.

SIN(X)	Sine of X (X is angle in radians)
COS(X)	Cosine of X (X is angle in radians)
TAN(X)	Tangent of X (X is angle in radians)
ATN(X)	Angle in radians whose tangent is X (arctangent)

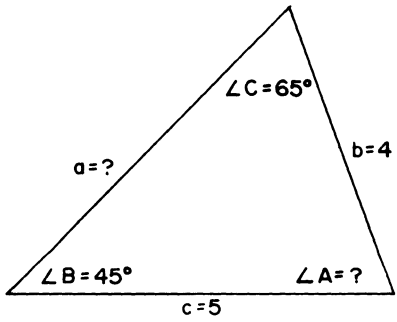
These functions are demonstrated in the program in Figure 4-4.

```
*LIST)
0010 REM :: DEMO OF TRIG CALCULATIONS
0020 REM :: P CONVERTS FR DEGREES TO RADIANS
0030 LET P=3.14159/180
0040 REM :: X IS 30 DEGREES IN RADIANS
0050 LET X=30*P
0060 PRINT SIN(X)
0070 PRINT COS(X)
0080 LET T=TAN(X)
0090 PRINT T
0100 REM :: ATN(T) IS RADIANS
0110 PRINT ATN(T)
0120 REM :: ATN(T)/P IS DEGREES
0130 PRINT ATN(T)/P
0140 END

*RUN)
 .5
 .866026
 .57735
 .523599
 30

END AT 0140
*
```

Exercise 4-4. You can use the following formula, derived from the law of cosines, to calculate the unknown side of a triangle given two sides and their opposits angles:

$$a = b*\cos C + c*\cos B.$$

Given a triangle with side b = 4, side c = 5, angle B = 45^{o}, and angle C = 65^{o}, solve for side a.



NICE, NEAT OUTPUT

So far, we've been using PRINT without a complete explanation of how it works. We have used commas and semicolons as separators in examples; maybe you have discovered that the format of your output varies with your punctuation. At this point, we have included the rules about PRINT punctuation, so you may choose the format of your printed output.

1. If a PRINT statement contains more than one item you must separate these items by a comma (,) or a semicolon (;).

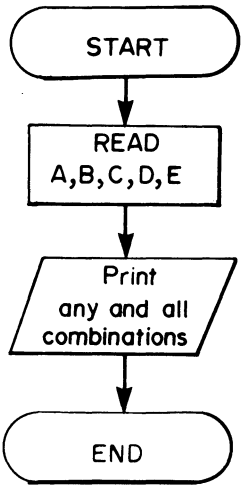
2. The print line on a terminal is divided into five print zones of 14 spaces each. If a comma (,) separates items in the PRINT statement, BASIC will print the next item in the leftmost position of the next printing zone. If no more print zones are available on a current line BASIC moves the item to the next line.

3. If a semicolon (;) separates items in the PRINT statement, BASIC will print the next item at the next character position. BASIC reserves a space before any positive number, and prints a minus sign before any negative number. BASIC also reserves one space after each number.

4. When BASIC prints the last item in a PRINT list, it outputs a carriage return/line feed combination unless a comma or semicolon follows the last item in the list. This carriage return/line feed will take you to the next line. If you punctuate the last item BASIC will not output the carriage return/line feed - it will print the next PRINT item on the same line according to the comma or semicolon punctuation.

5. A PRINT statement without print items or punctuation causes BASIC to output a carriage return/line feed combination. This PRINT statement will either complete a previous PRINT statement ending with a comma or semicolon, or will generate a blank line.

Examine the program in Figure 4-5 and decide how the PRINT statement affects its output.



SD-00272

Figure 4-5. The PRINT Statement.

```
*LIST)
0010 REM ** DEMO OF PRINT STATEMENT **
0020 READ A,B,C,D,E
0030 PRINT A,B,C
0040 PRINT D;E;A;B
0050 PRINT
0060 PRINT A,B,
0070 PRINT C
0080 PRINT A;B;
0090 PRINT C
0100 PRINT
0110 PRINT A,B,
0120 PRINT C;D;
0130 PRINT
0140 PRINT "A AND B FOLLOW: ",A,B
0150 PRINT "C; D AND E ARE NEXT: ";C;D;E
0160 PRINT
0170 PRINT A+B/C*D
0180 PRINT "D+E EQUALS ",D+E
0190 PRINT "A+B EQUALS ";A+B
0200 DATA 1,2,3,4,5

*RUN)
 1             2             3
 4   5   1   2

 1             2             3
 1   2   3

 1             2             3   4
A AND B FOLLOW:              1                   2
C; D AND E ARE NEXT:   3   4   5

 3.66667
D+E EQUALS      9
A+B EQUALS   3

END AT 0200
*
```

The comma at the end of line 60 tells BASIC not to output a carriage return and to print the C from line 70 on the same output line as the A & B from line 60.

Exercise 4-5. Study the program in Figure 4-6 and see if you can predict its output. Run the program and compare the computer's output with your own.

```
*LIST)
0010 REM ##### PRINT ME
0020 PRINT A,B,C
0030 READ X,Y,Z
0040 PRINT X;
0050 PRINT Y;
0060 PRINT Z
0070 PRINT
0080 PRINT "PLAYING COMPUTER IS FUN"
0090 PRINT 7,
0100 PRINT "6,5",4,
0110 PRINT
0120 PRINT "I CAN COUNT BACKWARDS",
0130 PRINT 3;2;1
0140 PRINT "THE END"
0150 DATA 5,7,9
0160 END

*
```

Figure 4-6. Program for Exercise 4-5.

END OF CHAPTER

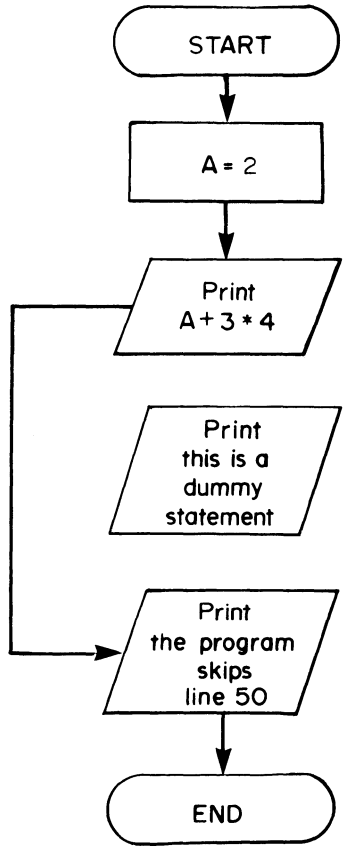
CHAPTER 5
DECISIONS AND LOOPS

Thus far, BASIC has run your program statements from top to bottom - from the lowest statement number in step to the highest. As you begin to write more complex programs, you will often find this order inefficient. This chapter describes ways to alter the flow of execution within your programs.

NON-NUMERICAL ORDER

As you write your program, you can direct BASIC to other statements within it. The GOTO statements alters the normal flow of execution by explicitly directing the program to some specific line number.

In Figure 5-1, the GOTO statement directs BASIC to skip a line. In the flowchart, the arrows show control bypassing the procedure box which contains "PRINT THIS IS A DUMMY STATEMENT". The arrows represent the GOTO statement in line 40, which directs control to line 60. BASIC never executes line 50.




```
*LIST)
0010 REM - GOTO STATEMENT
0020 LET A=2
0030 PRINT A+3*4
0040 GOTO 0060
0050 PRINT "THIS IS A DUMMY STATEMENT"
0060 PRINT "THE PROGRAM SKIPS LINE 50"
0070 END

*RUN)
 14
THE PROGRAM SKIPS LINE 50

END AT 0070
*
```

SD-00273

Figure 5-1. The GOTO Statement.

Exercise 5-1. Show what the computer will print when it executes the program in Figure 5-2.

```
*LIST)
0010 REM - FOLLOWING GOTO
0020 PRINT "DEMO OF GOTO"
0030 GOTO 0080
0040 PRINT "HAVE A GOOD DAY!"
0050 GOTO 0130
0060 PRINT "REALLY SKIPS AROUND, ";
0070 GOTO 0110
0080 PRINT
0090 PRINT "THIS PROGRAM ";
0100 GOTO 0060
0110 PRINT "DOESN'T IT?"
0120 GOTO 0040
0130 END
```

Figure 5-2. Going Round in Circles.

STOP OR END?

Until now, we have used the END statement to show the end of our programs; the END statement has been the last one BASIC reads. Yet as you redirect control, your bottom statement will often lead back into the program; the logical end will be somewhere in the middle. To separate the logical end of a program from its physical end, BASIC includes the STOP statement. Whenever the logical and physical ends of your program differ, you should use STOP for the logical end and reserve END for the physical end. Either statement will work - the two are functionally identical in Data General's BASIC - but both your flow charts and programs will be clearer if you distinguish the logical end from the physical end. We have made this distinction in Figure 5-3.

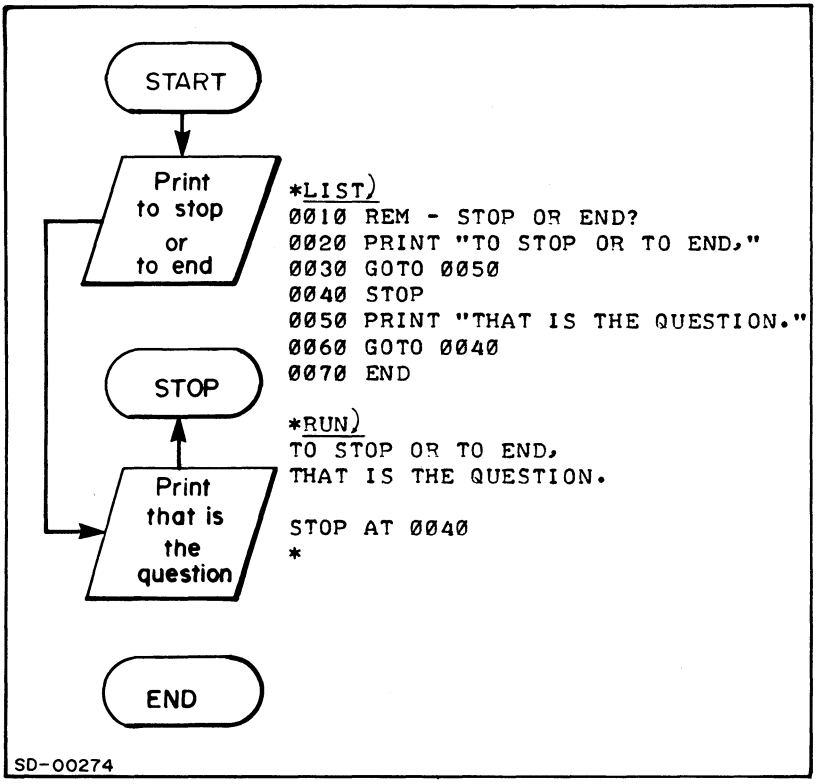


Figure 5-3. STOP and END Statements.

These last two program examples have really been jumping around, and while hopping and skipping may be fun, they don't represent efficient coding. In the next section, you'll learn an important new statement to use with GOTO - and some good reasons for applying STOP and END properly.

WHAT IF?

Many times a programmer would like to perform some calculation in one case and another calculation in another case. Maybe you are figuring a company payroll and want to add overtime compensation if an employee has worked over 40 hours. You can use the IF statement to test this condition (is hours worked over 40?) and add compensation IF the answer is yes.

A test condition implies a choice. The IF statement tests an expression and gives BASIC directions to follow if that expression is true. These directions follow the keyword THEN. An IF statement for the overtime program above is:

```
100  IF H>40 THEN GOTO 150
```

where H is the number of hours worked. The code at line 150 would contain the routine to figure the overtime.

Did you recognize the greater than sign (>) in line 100? BASIC includes several signs for testing relationships between numbers, called relational operators. Table 5-1 describes these relational operators.

Table 5-1. Relational Operators

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

Do you remember the discussion of flow charts in Chapter 3? The diamond-shaped decision box represented a test condition in the form of a "yes" or "no" question. A "yes" answer routed program flow one way, a "no" answer another way. Whenever your flow chart contains a decision box, you will use the BASIC IF statement.

Within your program, BASIC evaluates the relational expression following IF (H>40), and if this expression is true (if the answer to your question is "yes") BASIC will go to the statement following THEN (GOTO 150). If that expression is not true (if the answer to your question is "no"), BASIC will go to the statement which follows the IF.

Almost all BASIC statements are valid after the keyword THEN. You can use PRINT, LET or GOTO depending on what you want the program to do. Two exceptions we will note here are the END statement, though you can use THEN STOP, and FOR or NEXT statements which we'll tell you about later.

Suppose we show you an example. We will input a number N and test the relationship of some number A to N. If A is less than N, THEN we will print a message saying "A is less than N", add 2 to A and test the relationship again. If A is not less than N we will stop. The question in the flow chart decision box will be "Is A less than N?"

Figure 5-4 contains the flow chart and program for this problem. Notice the decision box in the flow chart and the GOTO in line 40 of the program.

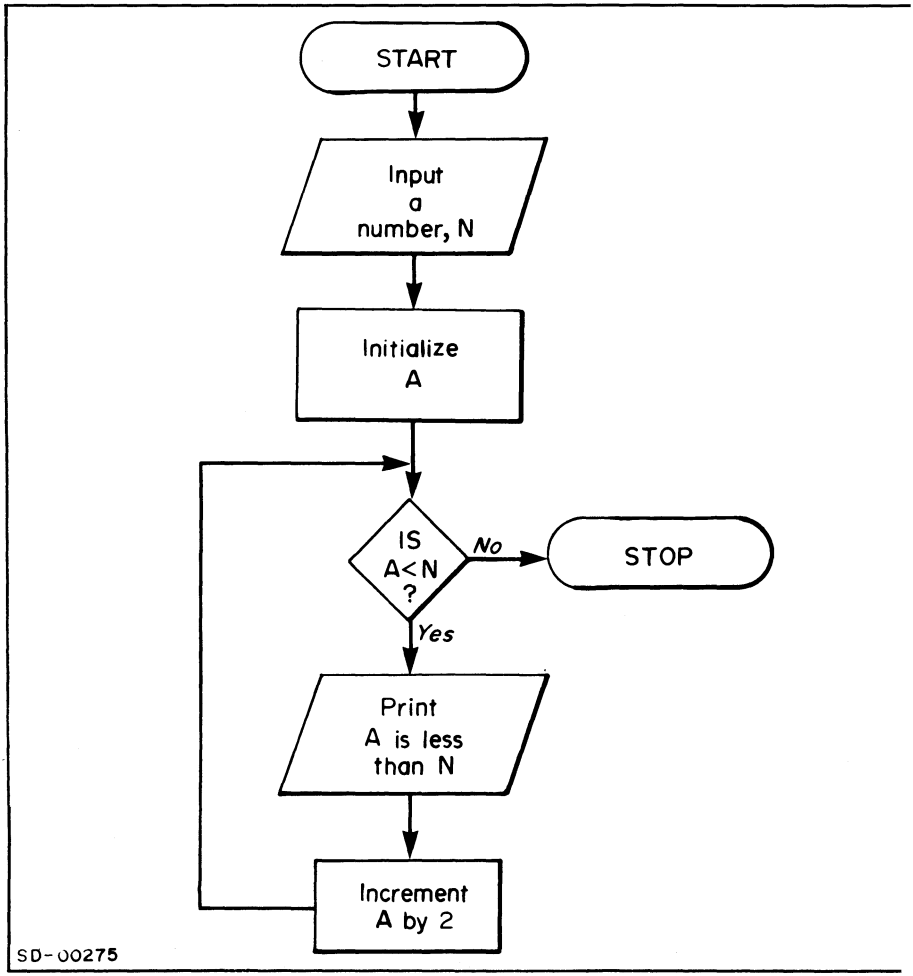


Figure 5-4. IF Statement.

Exercise 5-2. Dig out the program you wrote for Exercise 4-3 to figure a simple salary. Update the program to check for overtime. Calculate the first 40 hours at the given wage and any hours over 40 at time and a half (1 1/2 times the given wage). Print the employee number, hours worked, pay for first 40 hours, overtime pay and total salary.

Exercise 5-3. Write a program which asks you to INPUT 2 numbers, X and Y, and print their relationship (X is less than Y, X is greater than Y, or X is equal to Y).

```
*LIST)
0010 REM - IF...THEN DECISION
0020 INPUT "TYPE A NUMBER: ",N
0030 LET A=0
0040 IF A<N THEN GOTO 0060
0050 STOP
0060 PRINT A;" IS LESS THAN ";N
0070 LET A=A+2
0080 GOTO 0040
0090 END

*RUN)
TYPE A NUMBER: 9
 0  IS LESS THAN  9
 2  IS LESS THAN  9
 4  IS LESS THAN  9
 6  IS LESS THAN  9
 8  IS LESS THAN  9

STOP AT 0050
*
```

The term **initialize** has several meanings in the computer world. We use it here to define and set a counter for one program's use.

Numeric Expression

You may follow IF with a numeric variable (or **numeric expression**) in place of a relational expression. Whenever BASIC encounters a numeric expression after an IF, it checks that number against zero. If the number equals zero, the statement is false, and control goes to the next line in your program. If the number is not zero, the statement is true, and BASIC will follow the instructions after THEN.

```
0010 LET N=0
0020 IF N THEN GOTO 0070          Statement false, goto 30.
0030 LET N=10
0040 IF N THEN GOTO 0070          Statement true, goto 70.
```

The program in Figure 5-5 uses a numeric IF expression to test whether N equals zero. The program continues around in circles, or **loops**, until N equals zero.

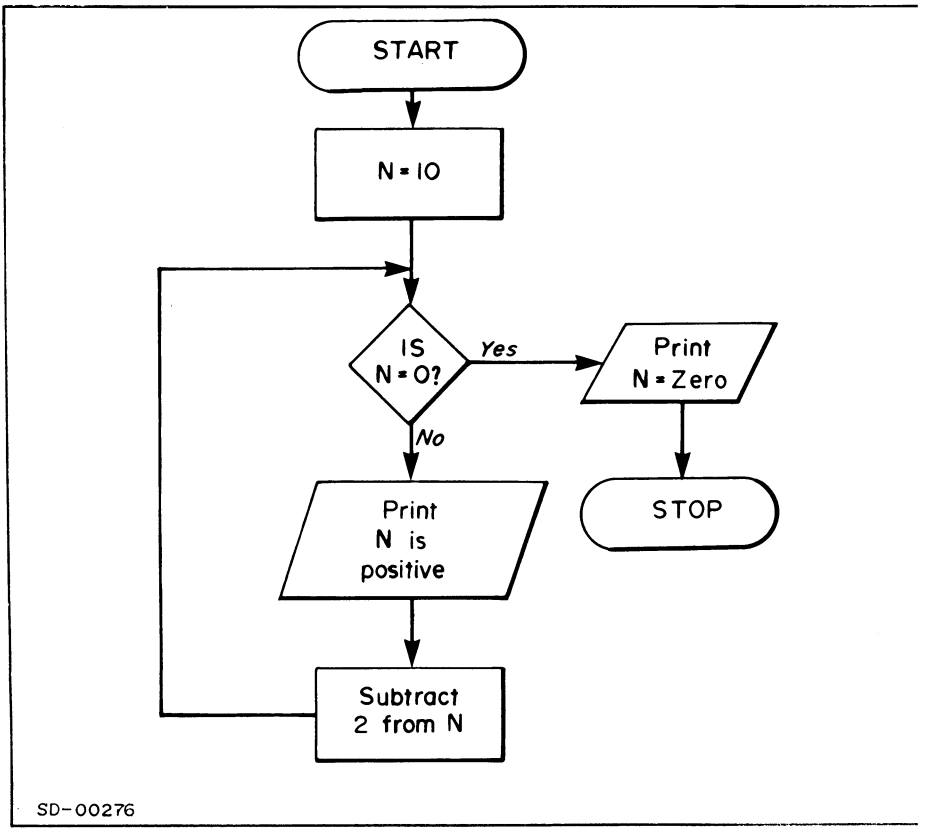


Figure 5-5. Numeric IF Statement.

You must be careful using the numeric test. If N had been set to 9 in line 20 of Figure 5-5, the program would have continued looping indefinitely because N would never equal 0. If you suspect your program is in an infinite loop, or if you want to stop it for some other reason, press the ESCape key. An ESCape will stop your program; BASIC will print the line number currently executing and an asterisk prompt. You may then edit, revise, or RUN the program again.

Exercise 5-4. The factorial of a number is that number, N, times (N-1) times (N-1)-1 down to 1 (not 0). We denote a factorial with an exclamation point (!). So 5! = 5*4*3*2*1. Write a program to input N and find N!.

```
*LIST)
0010 REM (IF WITH A NUMERIC EXPRESSION)
0020 LET N=10
0030 IF N THEN GOTO 0060
0040 PRINT " N  IS ZERO"
0050 STOP
0060 PRINT N;" IS POSITIVE"
0070 LET N=N-2
0080 GOTO 0030
0090 END

*RUN)
 10  IS POSITIVE
 8  IS POSITIVE
 6  IS POSITIVE
 4  IS POSITIVE
 2  IS POSITIVE
 N  IS ZERO

STOP AT 0050
*
```

Getting Complicated

So far, we have used fairly simple examples so you could learn the construction of BASIC programs. Figure 5-6 contains a more

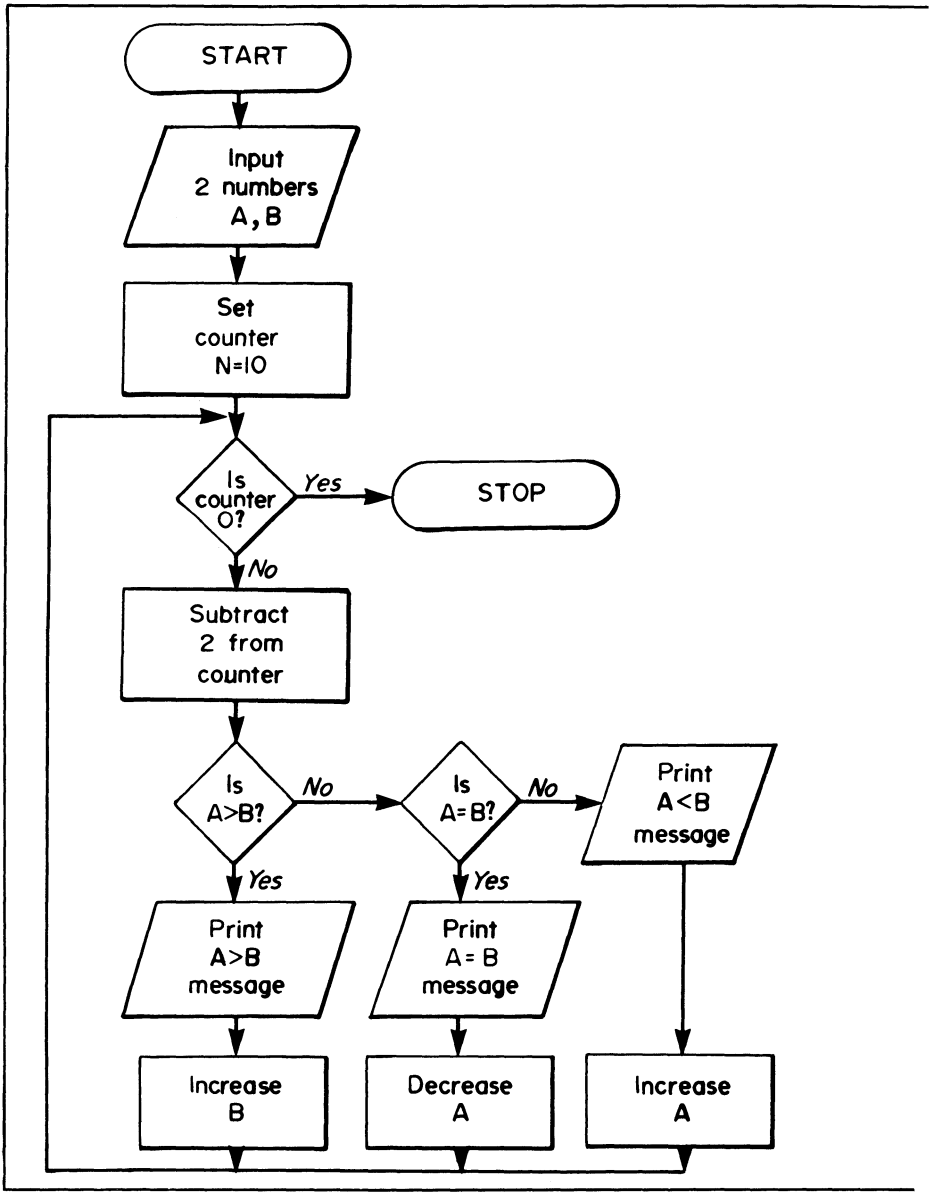


Figure 5-6. Number Comparison Program.

complex program using both IF relationships. The program compares 2 numbers, changes the value of the lower and compares again.

```
*LIST)
0010 REM  COMPARE & CHANGE NUMBERS
0020 INPUT "TYPE VALUES FOR A AND B: ",A,B
0030 LET N=10
0040 IF N THEN GOTO 0060
0050 STOP
0060 LET N=N-2
0070 IF A>B THEN GOTO 0130
0080 IF A=B THEN GOTO 0170
0090 REM - A MUST BE LESS THAN B
0100 PRINT A;" IS LESS THAN ";B
0110 LET A=A+B/2
0120 GOTO 0040
0130 REM - A GREATER THAN B
0140 PRINT A;" IS GREATER THAN ";B
0150 LET B=B+A/2
0160 GOTO 0040
0170 REM - A EQUAL TO B
0180 PRINT A;" IS EQUAL TO ";B
0190 LET A=A/2
0200 GOTO 0040
0210 END

*RUN)
TYPE VALUES FOR A AND B: 7,4
 7  IS GREATER THAN  4
 7  IS LESS THAN  7.5
 10.75  IS GREATER THAN  7.5
 10.75  IS LESS THAN  12.875
 17.1875  IS GREATER THAN  12.875

STOP AT 0050
*RUN)
TYPE VALUES FOR A AND B: 2,4
 2  IS LESS THAN  4
 4  IS EQUAL TO  4
 2  IS LESS THAN  4
 4  IS EQUAL TO  4
 2  IS LESS THAN  4

STOP AT 0050
*
```

SD-00277

Exercise 5-5. Write a program to input 10 numbers and print the largest. We have started a flow chart in Figure 5-7, though you will have to complete it.

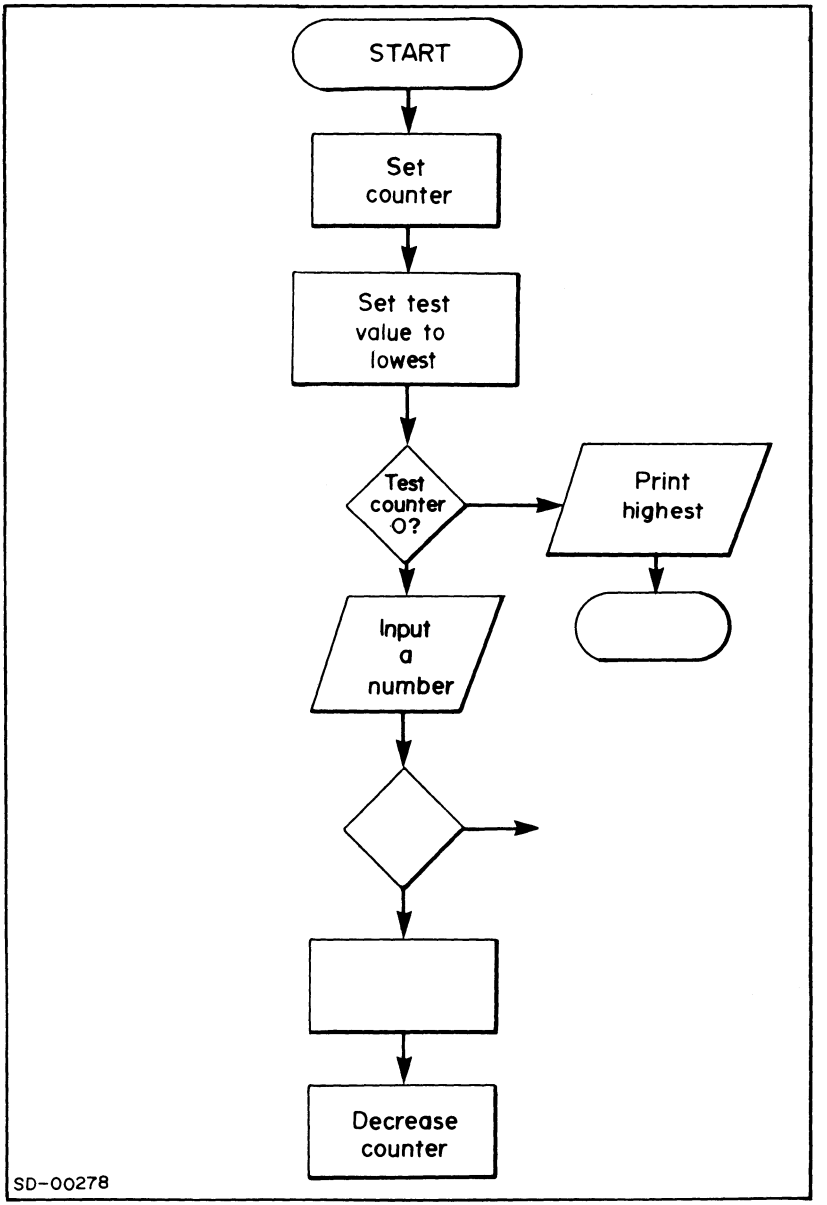


Figure 5-7. Incomplete Flow Chart for Exercise 5-5.

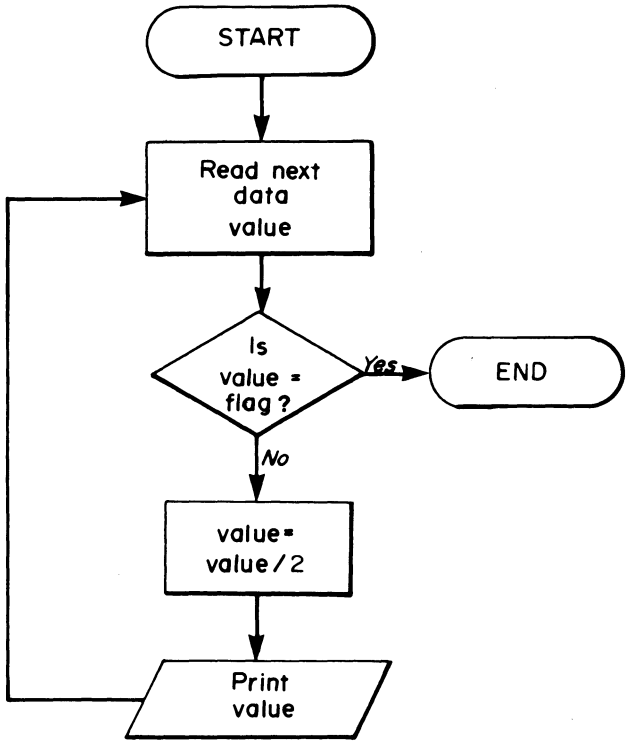
Flagging the End

You can also use the IF statement to tell BASIC that it has reached the end of a DATA list. The IF statement in line 30 of Figure 5-8 tests each item in the DATA list for a value out of the normal data range. We use this value as a **flag** to mark the end of the data.

On the first pass through this program the READ statement takes the first DATA value, 2. Since A is less than 9999 in line 30, BASIC divides 2 by 2 and prints 1. When control goes back to line 20 BASIC reads the second DATA value, 4. This process continues until BASIC finds the 9999 flag and the program STOPS at line 30.

With this technique, you can write your programs to READ different DATA statements each time they run, or to READ a sequence of DATA statements for one run. You must flag the last DATA value; if you don't, BASIC will return an error message when it runs out of values.

Exercise 5-6. You can also use a flag value to signal the end of an INPUT list. Input a list of numbers and end the program with a zero flag. For each number use the integer function to print its fractional part. Print negative sign if the original number is negative. If 4.38 is input, your program should print .38. If -6.24 is input, your program should print -.24. If 1000 is input, your program should stop.




```
*LIST)
0010 REM **9999 FLAG**
0020 READ A
0030 IF A<9999 THEN GOTO 0050
0040 STOP
0050 LET A=A/2
0060 PRINT A
0070 GOTO 0020
0080 DATA 2,4,6,8,9999
0090 END

*RUN)
 1
 2
 3
 4

STOP AT 0040
*
```

SD-00279

Figure 5-8. Program Using a Flag Value.

RANDOMIZE

Thus far, BASIC has given you nothing original for all your input; it has simply done what you told it to do. The RANDOMIZE functions do generate something for you - random numbers between zero and one.

The RND(0) function returns a psuedo-random number between 0 and 1 each time you call it. BASIC generates the same sequence of random numbers each time you use RND(0), unless you add the RANDOMIZE statement. The RANDOMIZE statement resets BASIC's random number generator to produce a different sequence for each run. These numbers allow you to play many games of chance with BASIC.

If you write your program using the RND(0) function, you can debug it with the same set of random numbers. Once your program runs correctly, insert the RANDOMIZE statement to produce a unique sequence of numbers for each run.

The program in Figure 5-9 simulates a coin game. The program generates one thousand random numbers and counts them as heads if greater than .5 and tails if equal to or less than .5. It prints the total number of heads and tails thrown.

Exercise 5-7. Write a program to generate a random integer between 1 and 100. Have the program ask you to guess a number and tell you whether your guess is higher or lower than the computer's number. You should be able to guess the computer's number in 7 tries.

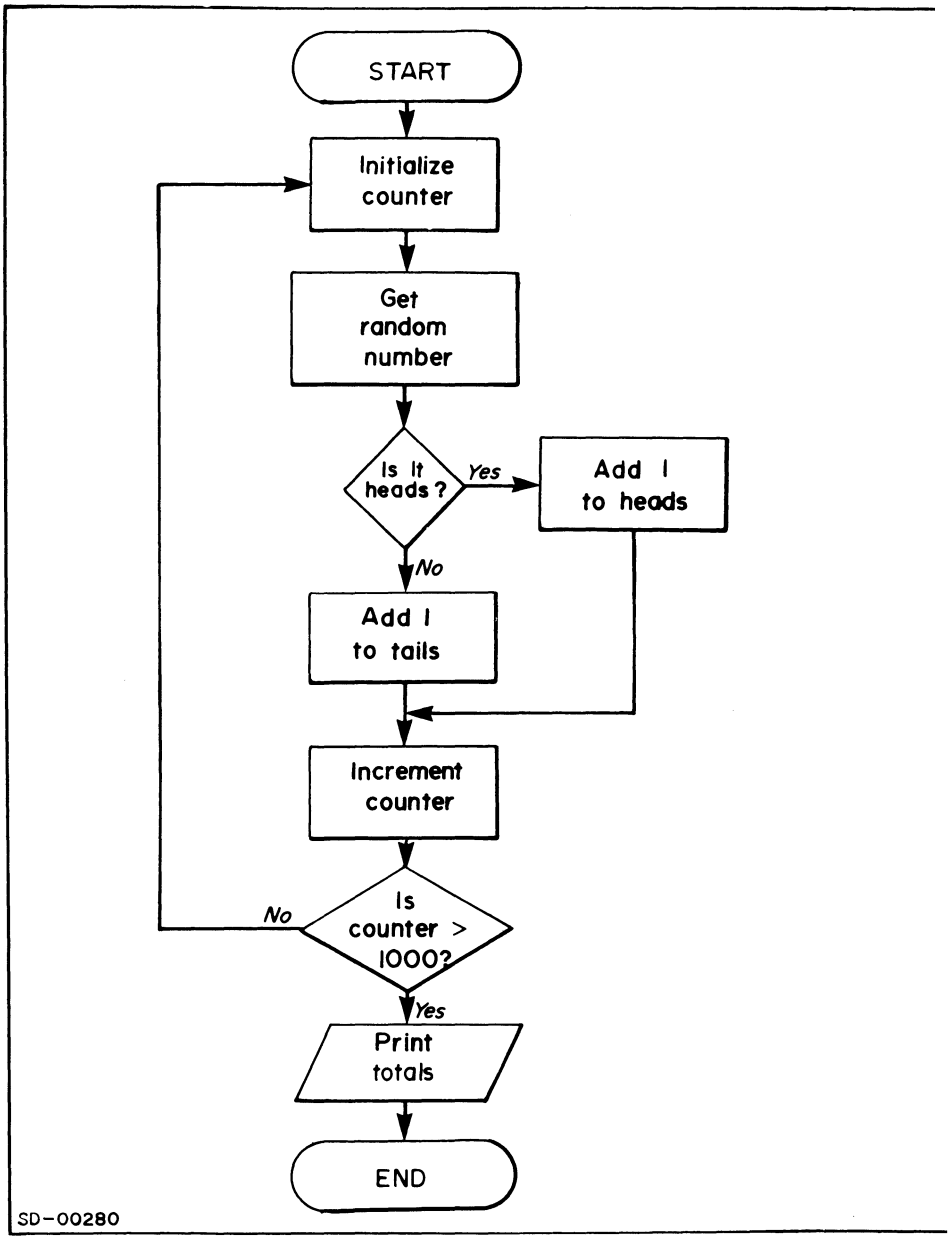


Figure 5-9. Heads or Tails?

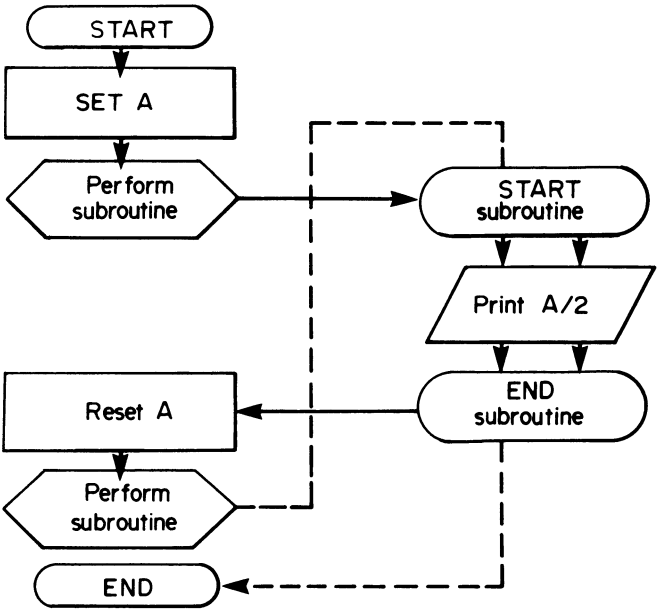
```
*LIST)
0010 REM - COIN GAME
0020 LET C=1
0030 RANDOMIZE
0040 LET N=RND(0)
0050 IF N>.5 THEN GOTO 0090
0060 REM - ADD 1 TO TAILS
0070 LET T=T+1
0080 GOTO 0110
0090 REM - ADD 1 TO HEADS
0100 LET H=H+1
0110 LET C=C+1
0120 IF C<1000 THEN GOTO 0040
0130 PRINT "OUT OF 1000 THROWS,"
0140 PRINT "     ";H;"WERE HEADS,"
0150 PRINT "     ";T;"WERE TAILS."
0160 END

*RUN)
OUT OF 1000 THROWS,
     493 WERE HEADS,
     506 WERE TAILS.

END AT 0160
*
```

USING A SUBROUTINE

As you create more complex programs, maybe you've noticed that certain sequences of statements appear more than once. These sequences may perform calculations, or compare variables- in any case, they are identical, and you've had to write them more than once.

You can code these sequences as **subroutines.** In BASIC, you can write a subroutine once, and return to it whenever you want. After you have written your subroutine statements, and given them line numbers, use the GOSUB statement to direct program control to them, and the RETURN statement to bring control back to the statement below GOSUB. Subroutines are the building blocks of large programs; whether they are long or short, they save time, and help you to see complex programs as an assembly of simple units. Remember that each GOSUB must precede its RETURN, and that BASIC will RETURN to the statement after the GOSUB.

The flow chart for the program in Figure 5-10 includes a new symbol which transfers control to the beginning of a subroutine. You can omit the lines going from the program to the subroutine when you understand the subroutine procedure.




```
*LIST)
0010 REM - SUBROUTINE PROGRAM
0020 LET A=6
0030 GOSUB 0070
0040 LET A=10
0050 GOSUB 0070
0060 STOP
0070 REM - SUBROUTINE
0080 PRINT A/2
0090 RETURN
0100 END

*RUN)
 3
 5

STOP AT 0060
*
```

SD-00399

Figure 5-10. Subroutine.

Exercise 5-8. Write a program that READS a number A and uses a subroutine to stop if the number is less than zero or greater than 99. If A is within that range the program should subtract A from 100, print the answer, return, read the next DATA item and test again.

Exercise 5-9. Write a program to simulate a dice game. For each roll of the dice, use a subroutine which generates 2 random integers between 1 and 6. On the first toss you win with a total of 7 and the computer wins with a total of 12. Any other sum becomes your point. You may continue throwing, trying to match this point. If you roll a 7 while trying for a point, the computer wins.

Keep a tally of games won and lost. For a real dice game, your first roll wins with a 7 or 11, and the computer's first roll wins with 2, 3, or 12. Modify the program so that 2 or more players can compete. Have fun. Who needs Vegas?

END OF CHAPTER

CHAPTER 6

FOR/NEXT LOOPS

LOOP USING IF

In the next section we introduce a loop called a FOR/NEXT loop. The program in Figure 6-1 uses a familiar IF loop. It increments a variable, A, by 2 and prints A five times. The variable I counts the number of loops executed, and stops the program after five have been executed. We will code a corresponding program using a FOR/NEXT loop to show you the relationship between the two loops.

Exercise 6-1. The next section introduces a set of statements and demonstrates a different way to code this program. Can you think of other ways using the statements you already know?

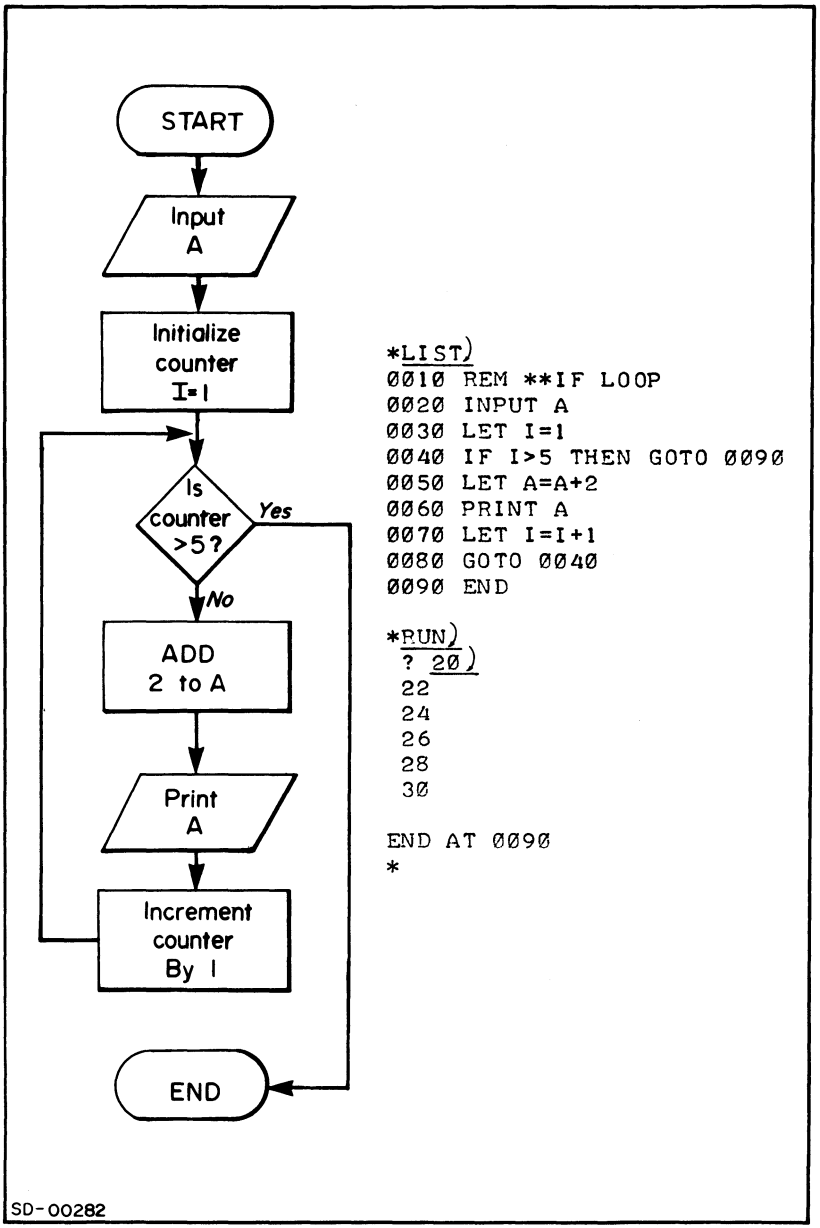


Figure 6-1. IF Loop.

THE FOR AND NEXT STATEMENTS

The FOR and NEXT statements mark the beginning and end of a program loop. FOR assigns a range to a variable and tells BASIC how many times to execute the loop; NEXT directs control back to FOR. The two statements allow you to loop as many times as you want.

FOR/NEXT loops extend the usefulness of BASIC; they are convenient, self-terminating and use fewer program statements than any other loop.

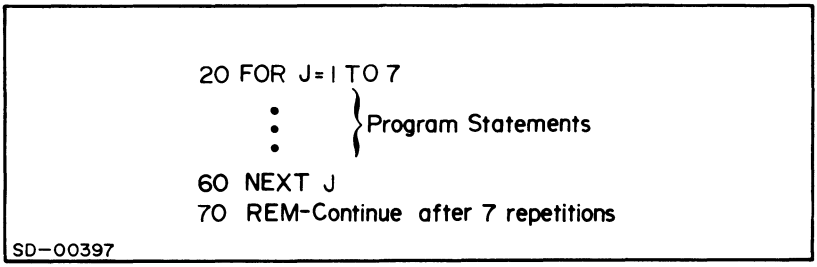


Figure 6-2. FOR and NEXT Statements.

Assume that BASIC is reading the program in Figure 6-2 for the first time. At line 20, BASIC will initialize J to the first value in J's range (1). J is then tested against the last value in the range, 7. J is less than 7, so BASIC will execute the statements between line 20 and line 60. At line 60, BASIC adds 1 to J, and loops back to test the new J against 7. After six more loops, J exceeds 7; control passes from line 20, where the test occurred, to the line following the NEXT statement, line 70.

Figure 6-3 shows the FOR/NEXT version of the IF program in Figure 6-1. The dotted lines in the flowchart show the logic which the FOR/NEXT statements perform but the programmer does not code. When you understand the logic of the FOR/NEXT loop you can omit the dotted sections from your flowcharts.

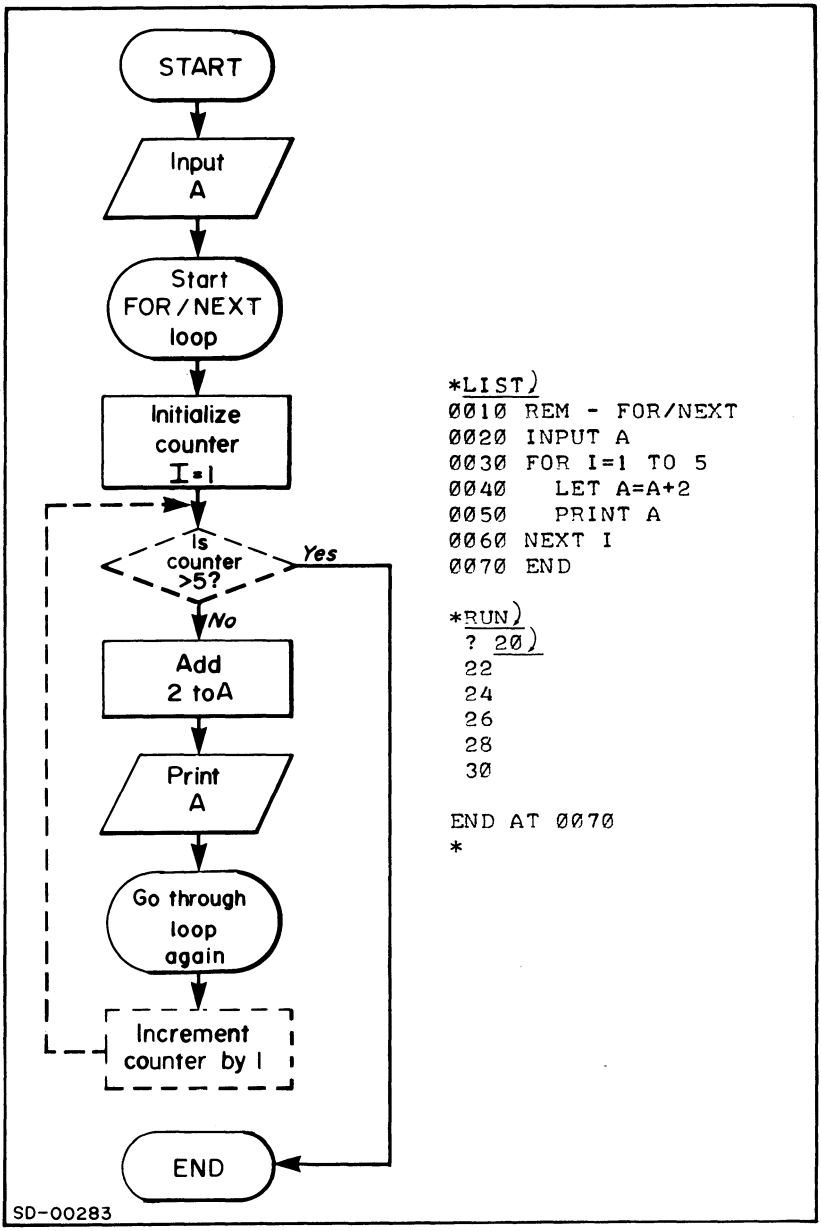


Figure 6-3. FOR/NEXT Loop.

To distinguish FOR/NEXT loops from the rest of a program, BASIC always indents them (lines 40 and 50). Note that the FOR/NEXT loop uses 7 statements for this program and the IF loop uses 9. Remember that a FOR statement must always have a corresponding NEXT statement to end the loop.

Explanations alone may not clarify FOR/NEXT loops; you must try some examples. Therefore, we have included more exercises than usual here and hope you will try them. The concept of FOR/NEXT loops is a little tough but once you understand it you may never use an IF loop again.

Exercise 6-2. Write a program that squares the integers from 1 to 100 and prints the integer and its square if the square is odd. Use a FOR/NEXT loop and the flow chart in Figure 6-4. The integer which you are squaring is the variable you are incrementing in your FOR/NEXT loop. Clue: the INT(X) function can help test for odd numbers.

Exercise 6-2A. A more complex variation of exercise 6-2 prints the number only if the integer in the tens column of the square is odd. The logic is tricky!

Exercise 6-3. Use a FOR/NEXT loop to write a BASIC program that will list the factors of a number (N) input from the terminal. A factor is a number which will produce N when multiplied by another factor. Use a FOR/NEXT loop to test the counter values from 1 to N.

Exercise 6-4. Write a program to balance your checkbook. Input the last balance and the number of checks written. Use a subroutine with a FOR/NEXT loop to total the amount of the checks. Subtract the check total and any service charge from the balance. Input the number and amount of deposits and add the deposit total to the balance. Print the current balance.

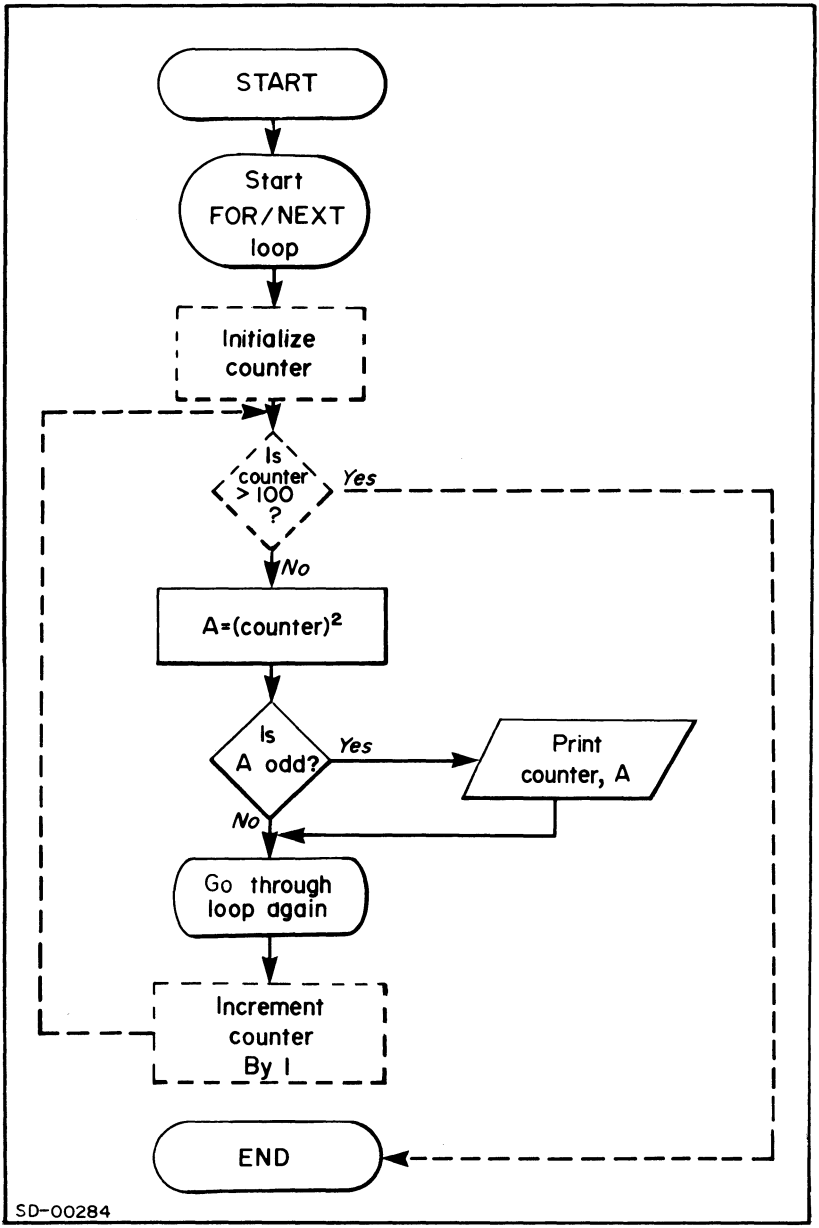


Figure 6-4. Flow chart for Exercise 6-2.

ADVANCED FUNCTIONS

Let's take a breather from FOR/NEXT loops to introduce the three functions in the program in Figure 6-5. If you haven't covered these functions in your math classes yet, you may skip them.

Square Root Function:

SQR(X) returns the value of the square root of X.

Exponential Function:

EXP(X) returns the value of e (2.71828) to the power of X.

Logarithm Function:

LOG(X) calculates the natural logarithm (base e) of X.

```
*LIST)
0010 REM '''ADVANCED MATH FUNCTIONS'''
0020 PRINT "SQR(25) = ";SQR(25)
0030 PRINT "EXP(1.5)= ";EXP(1.5)
0040 PRINT "LOG(959)= ";LOG(959)
0050 END

*RUN)
SQR(25) =  5
EXP(1.5)=  4.48169
LOG(959)=  6.86589

END AT 0050
*
```

Figure 6-5. Square Root, Exponential and Logarithm Functions.

Exercise 6-5. Check the accuracy of BASIC's SQR(X) function. Take the square root of each number from 1 to 25 and square it. Print the number, its square root, and square root squared.

Exercise 6-6. Write a program that will compare the product of 2 numbers, X and Y, with the exponent of the sum of their logarithms.

```
X*Y = Z    X1 = log(X)
           Y1 = log (Y)
           Z1 = X1 + Y1
           Z1 = EXP(Z1)
```

Z and Z1 should be very close.

NESTED FOR/NEXT LOOPS

Now, back to FOR/NEXT loops. You can nest FOR/NEXT loops within each other. An inner FOR/NEXT loop must be completely contained within the outer FOR/NEXT loop in which it is nested, as diagrammed in Figure 6-6.

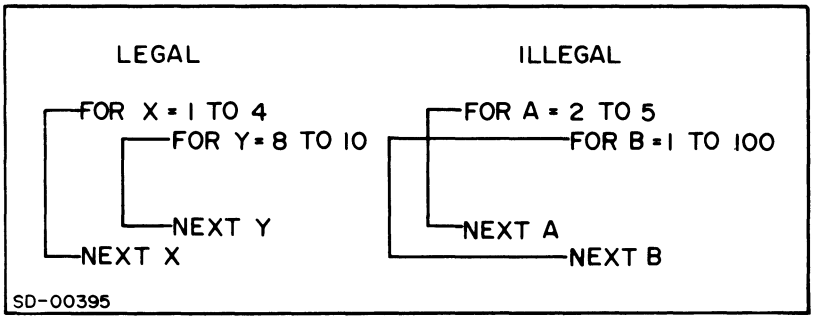


Figure 6-6. Legal and Illegal FOR/NEXT Nesting.

The program in Figure 6-7 prints the values of I and J within a nested FOR/NEXT loop.

The next chapter introduces a method of storing data and retrieving it using FOR/NEXT loops, both before unnested and nested. Get familiar with these loops before tackling Chapter 7.

Exercise 6-7. The flow chart with all its dotted lines can get very complicated. Use the flow chart without the dotted sections in Figure 6-8 to write a program which will print the multiplication tables up to 5 * 10.

Exercise 6-8. Revise the program you wrote for exercise 6-3 to list the factors of all the integers between 20 and 30. Use nested FOR/NEXT loops.

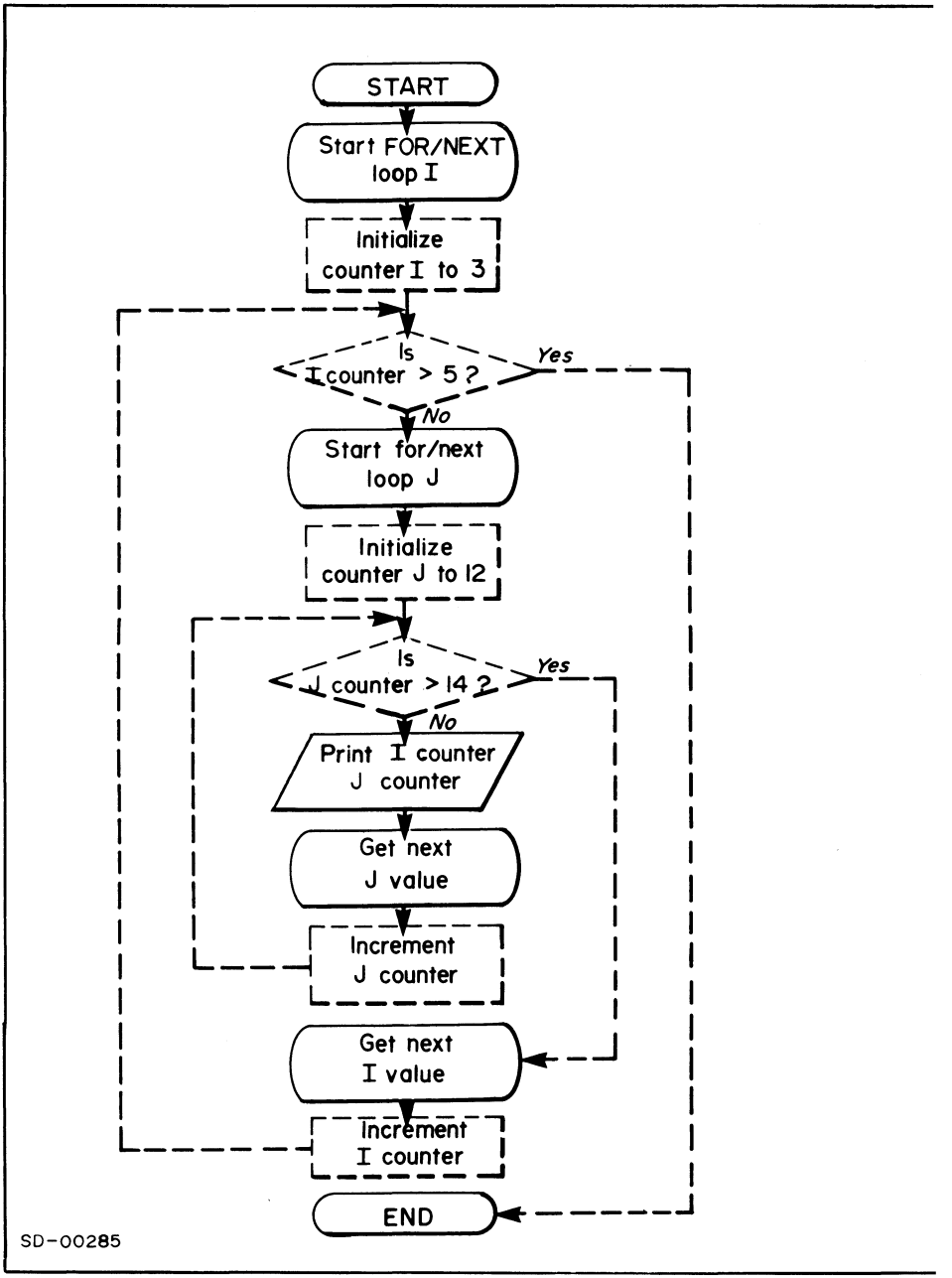


Figure 6-7. Nested FOR/NEXT loop.

```
*LIST)
0010 REMARKS - NESTED FOR/NEXT LOOP
0020 FOR I=3 TO 5
0030   FOR J=12 TO 14
0040     PRINT "I = ";I,
0050     PRINT "J = ";J
0060   NEXT J
0070 NEXT I
0080 END

*RUN)
I =  3          J =  12
I =  3          J =  13
I =  3          J =  14
I =  4          J =  12
I =  4          J =  13
I =  4          J =  14
I =  5          J =  12
I =  5          J =  13
I =  5          J =  14

END AT 0080
*
```

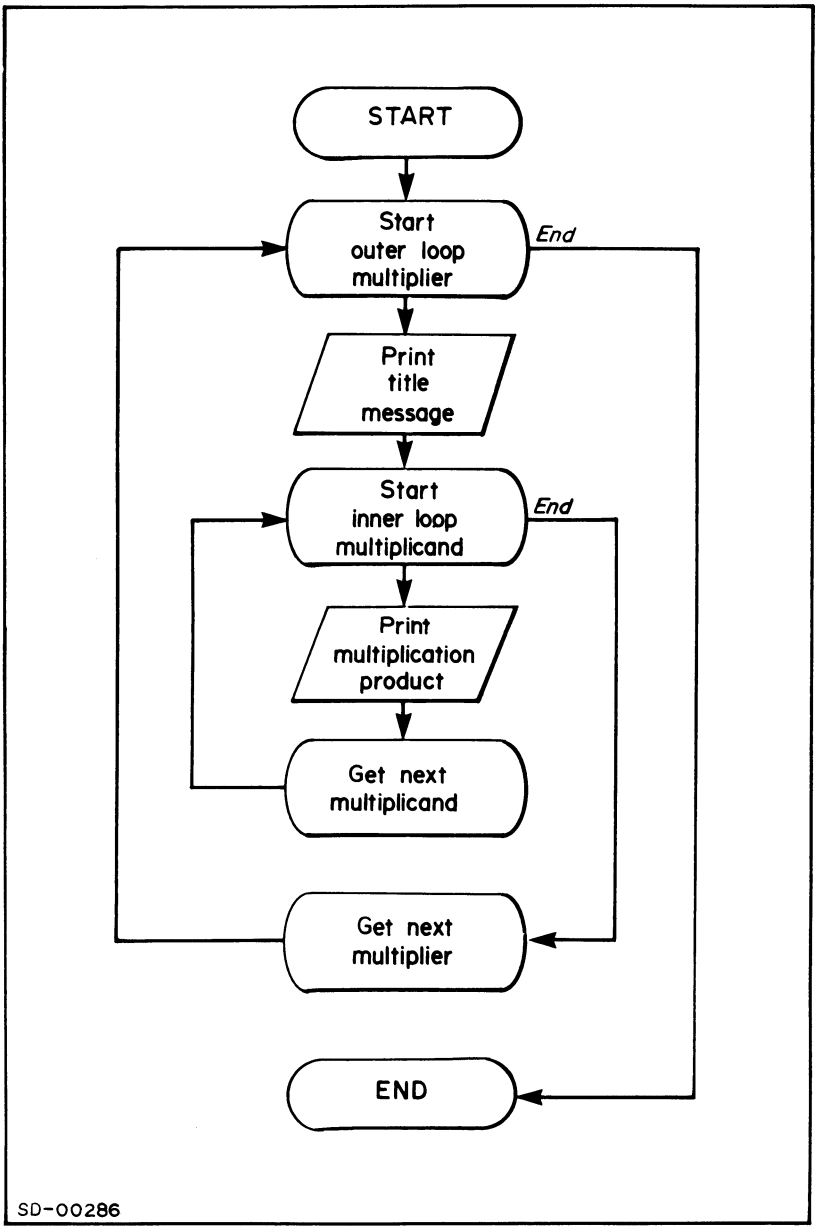


Figure 6-8. Flow Chart for Exercise 6-7.

END OF CHAPTER

CHAPTER 7

NUMERIC SUBSCRIPTING

ARRAYS

Thus far, you have stored single numbers in memory locations (mailboxes). BASIC also allows you to store tables of numbers in memory. These tables are called **arrays**; the numbers which they include are called **array elements.**

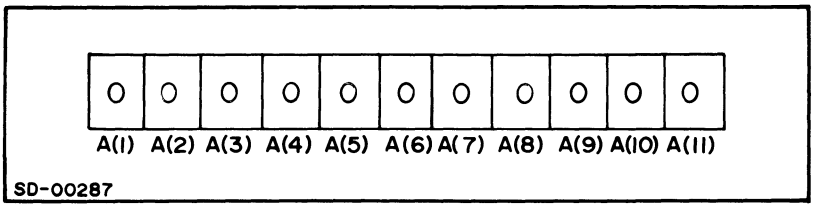


Figure 7-1. An Array.

Array A's box ranges from A(1) to A(11). (Some BASIC systems start numbering elements at 0.) You can place any number (or expression) you like in the boxes with the LET statement. You could LET A(1) equal 1399, -6, SQR(5), or 2/3.

To create an array, tell BASIC how many elements you want to use with the DIMension statement; DIM A(11) would create array A (Figure 7-1). Whenever you create an array, BASIC sets the value of each element to zero.

Figure 7-2 shows Array B, after program statement 10 created it, through line 50, after elements 2, 20, 23, 30 have received values.

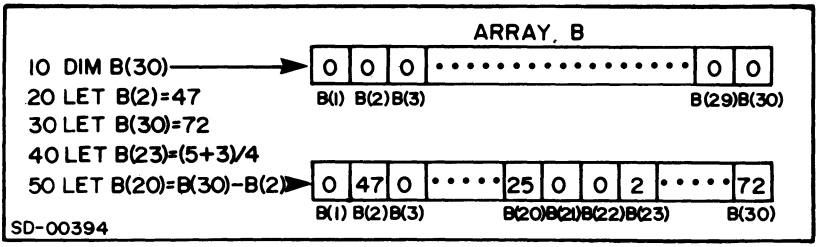


Figure 7-2. Program Statements and Array B.

As an example, the program in Figure 7-3 creates an array with an element for each of 6 employees. The program reads the employee's number (1 to 6) and hours worked. It then multiplies each number of hours by a fixed wage of $2.60 and places the resulting earnings in the employee's element of Array T. After it calculates all 6 wages it prints the elements in Array T. Notice how the variable J (the employee's number) in the FOR/ NEXT loop becomes the subscript for Array T.

FOR/NEXT variables and array subscripts each start at an initial value, get incremented and end at some final value; they really work well together, and we will use them often.

Exercise 7-1. Input an array, T, with 15 elements. Fill a second array, S, with the square roots of those elements, so S(1) = SQR(T(1)). Print array T and array S.

Exercise 7-2. Write a program that inputs an array of 10 elements. Find the highest and lowest element. Print the array and messages designating the highest and lowest elements.

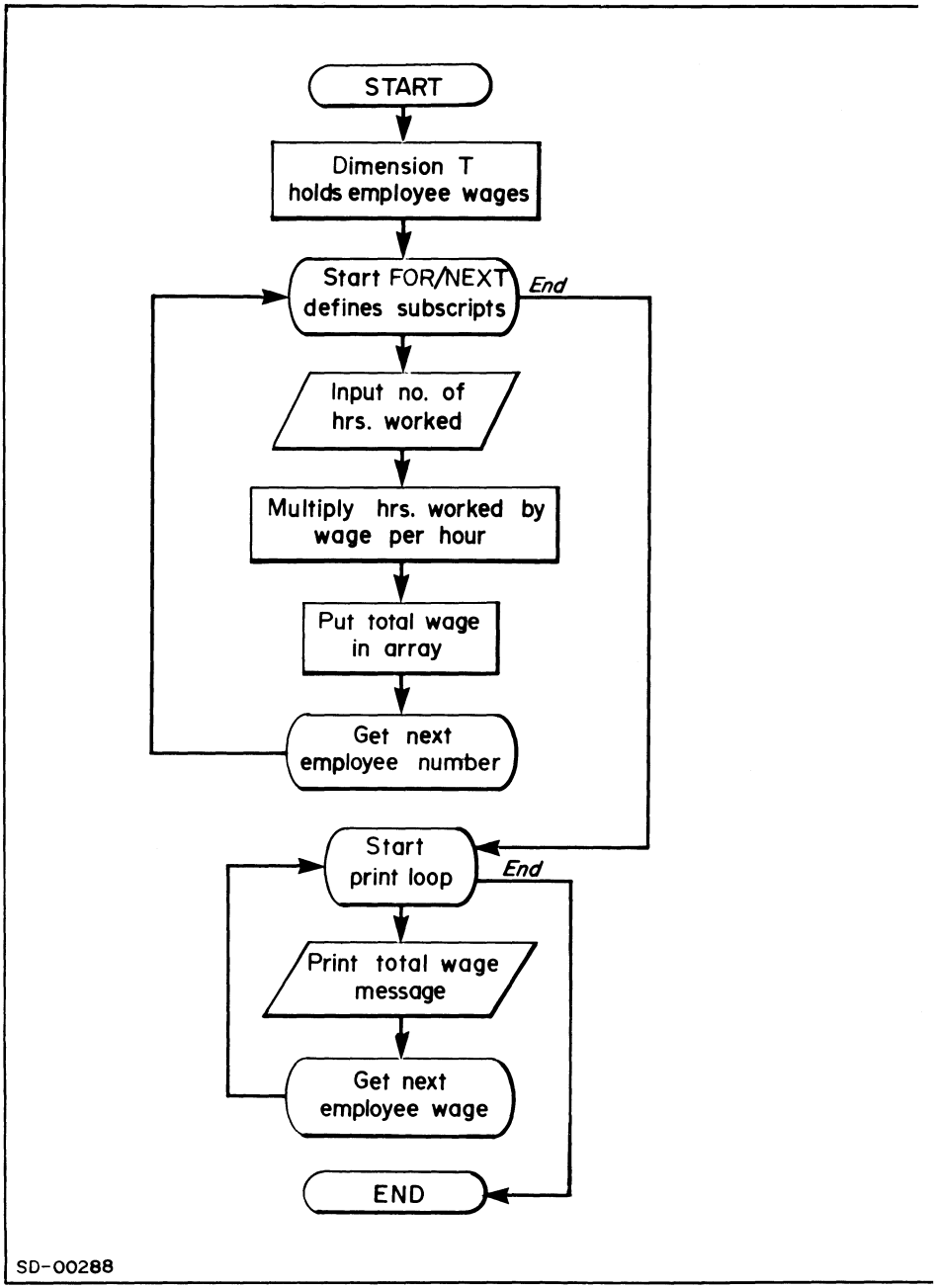


Figure 7-3. Array T Contains Employee Wages.

```
*LIST)
0010 REM ** FIGURES EMPLOYEE WAGES
0020 REM ** USES ARRAY
0030 DIM T[6]
0040 FOR J=1 TO 6
0050   PRINT "FOR EMPLOYEE ";J;"? "
0060   INPUT "TYPE HOURS WORKED: ",A
0070   PRINT
0080   LET T[J]=A*2.6
0090 NEXT J
0100 REM ** PRINT ARRAY
0110 FOR I=1 TO 6
0120   PRINT "EMPLOYEE #";I;" EARNED $";T[I]
0130 NEXT I
0140 END

*RUN
FOR EMPLOYEE  1 ?
TYPE HOURS WORKED: 40)

FOR EMPLOYEE  2 ?
TYPE HOURS WORKED: 35)

FOR EMPLOYEE  3 ?
TYPE HOURS WORKED: 42)

FOR EMPLOYEE  4 ?
TYPE HOURS WORKED: 10)

FOR EMPLOYEE  5 ?
TYPE HOURS WORKED: 38)

FOR EMPLOYEE  6 ?
TYPE HOURS WORKED: 20)

EMPLOYEE # 1  EARNED $ 104
EMPLOYEE # 2  EARNED $ 91
EMPLOYEE # 3  EARNED $ 109.2
EMPLOYEE # 4  EARNED $ 26
EMPLOYEE # 5  EARNED $ 98.8
EMPLOYEE # 6  EARNED $ 52

END AT 0140
*
```

ANOTHER ARRAY

The program in Figure 7-3 demonstrates the manipulation of array elements. The program totals the values of the array elements and stores the running total in the last array element.

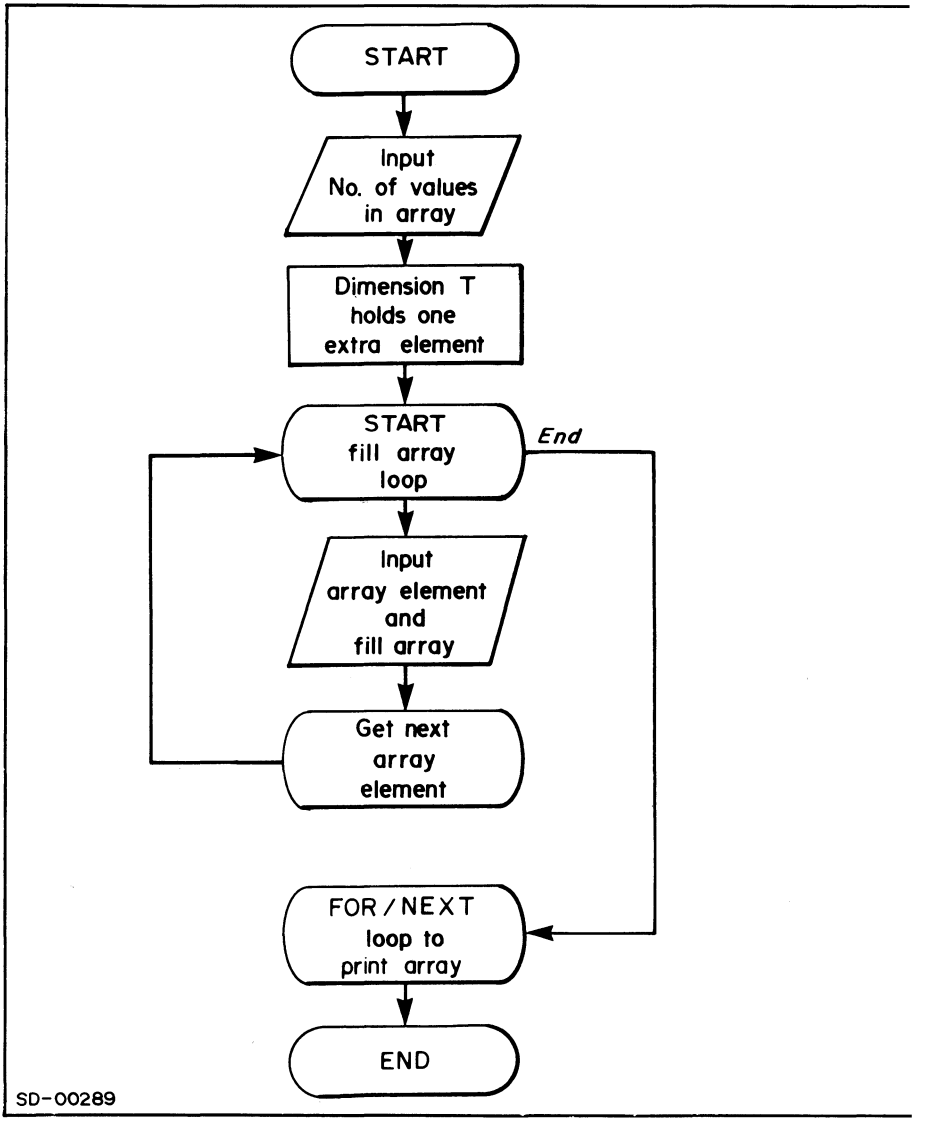
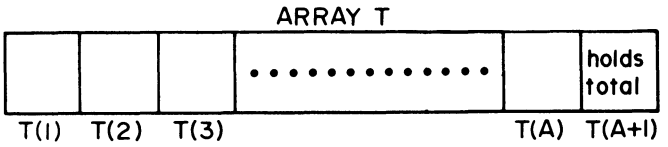


Figure 7-4. Program to Total Array T.

Exercise 7-3. Input array size and array elements from the terminal, sort the array in ascending order and print it. As you input each element, test it against those ahead of it, and move it to the proper position.




```
*LIST)
0010 REM - TOTALS ARRAY T
0020 INPUT "NUMBER OF VALUES TO BE ADDED: ",A
0030 DIM T[A+1]
0040 PRINT "TYPE NUMBERS TO BE ADDED:"
0050 FOR I=1 TO A
0060   INPUT T[I]
0070   LET T[A+1]=T[A+1]+T[I]
0080 NEXT I
0090 PRINT
0100 REM - FOR/NEXT PRINTS ARRAY
0110 FOR I=1 TO (A-1)
0120   PRINT T[I];"+";
0130 NEXT I
0140 PRINT T[A];" = ";T[A+1]
0150 END

*RUN)
NUMBER OF VALUES TO BE ADDED: 5)
TYPE NUMBERS TO BE ADDED:
 ? 10)
 ? 15)
 ? 30)
 ? 45)
 ? 72)

 10 + 15 + 30 + 45 + 72  =  172

END AT 0150
*
```

TWO DIMENSIONAL ARRAYS

Hang on, arrays are getting more complicated. A numeric array may have a second dimension! You've made arrays with rows; now we'll show you arrays with rows and columns.

For two-dimensional arrays, use two subscripts within parentheses in the DIM statement. The first number designates rows, the second columns. Therefore DIM D(4,3) in Figure 7-5 defines a two dimensional storage area in memory with 4 rows and 3 columns, all initialized to 0.

```
10 DIM D(4,3)
```

ARRAY D

row \ col.	1	2	3
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0

SD-00392

Figure 7-5. Two Dimensional Array.

To change the values of the elements within an array or to reference elements, always use the first subscript for the row number and second for the column number. Figure 7-6 contains the individual names for all the elements in Array D.

```
D(1,1)=10   D(1,2)=20   D(1,3)=30
D(2,1)=40   D(2,2)=50   D(2,3)=60
D(3,1)=70   D(3,2)=80   D(3,3)=90
D(4,1)=100  D(4,2)=110  D(4,3)=120
```

ARRAY D

row \ col.	1	2	3
1	10	20	30
2	40	50	60
3	70	80	90
4	100	110	120

SD-00393

Figure 7-6. Referencing Array Elements.

Now you are going to want to know why you'd ever need two dimensional arrays. Use these arrays for storing any data which you want to reach easily. Any table - tax tables, work rate tables, even multiplication tables - fits neatly into two dimensional arrays. You can hold the number of votes for the president and your senator in a two dimensional array. We will show you two examples and you can try some exercises.

The program in Figure 7-7 fills a 2 dimensional array with the sum of the subscripts of each element, so T(1, 2) = 1 + 2 = 3. Notice how we used the nested FOR/NEXT loop to manage the subscripts.

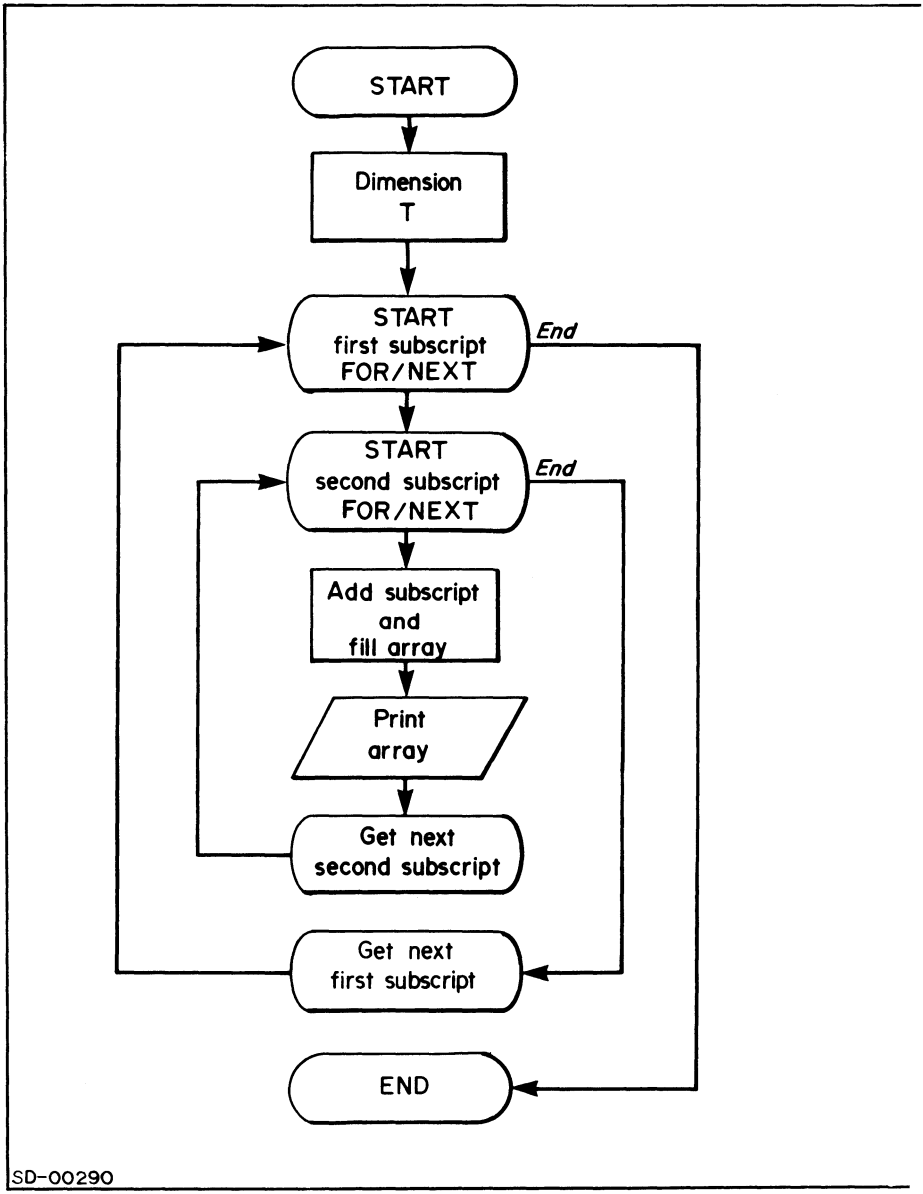


Figure 7-7. Program with Two Dimensional Array T

row \ col.	1	2
1	2	3
2	3	4
3	4	5

```
*LIST)
0010 REM ** 2-DIMENSIONAL ARRAY
0020 DIM T[3,2]
0030 FOR I=1 TO 3
0040   FOR J=1 TO 2
0050     LET T[I,J]=I+J
0060     PRINT "T(";I;",";J;") = ";T[I,J],
0070   NEXT J
0080   PRINT
0090 NEXT I
0100 END

*RUN)
T( 1 , 1 ) =  2              T( 1 , 2 ) =  3
T( 2 , 1 ) =  3              T( 2 , 2 ) =  4
T( 3 , 1 ) =  4              T( 3 , 2 ) =  5

END AT 0100
*
```

As another example the program in Figure 7-8 fills a 2 dimensional array with 4 rows and 3 columns. We input Variables A, B and C and the program multiplies them by the row subscripts.

row \ col.	1	2	3
1	1 × A	1 × B	1 × C
2	2 × A	2 × B	2 × C
3	3 × A	3 × B	3 × C
4	4 × A	4 × B	4 × C

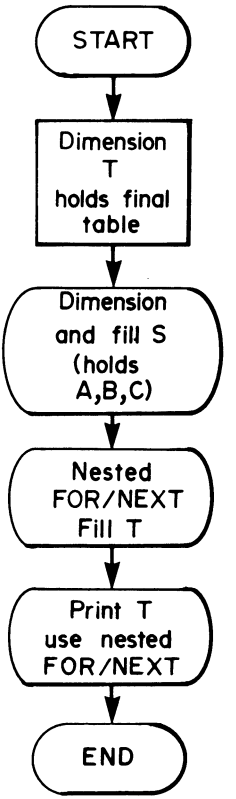


Figure 7-8. Two Dimensional Array Holds Multiplication Tables

```
*LIST)
0010 REM - MULTIPLICATION TABLE
0020 DIM T[4,3]
0030 REM - DIMENSION AND FILL ARRAY S
0040 DIM S[3]
0050 PRINT "TYPE A,B,C:"
0060 FOR I=1 TO 3
0070   INPUT S[I]
0080 NEXT I
0090 REM - NESTED LOOP FILLS ARRAY T
0100 FOR I=1 TO 4
0110   FOR J=1 TO 3
0120     LET T[I,J]=I*S[J]
0130   NEXT J
0140 NEXT I
0150 PRINT
0160 REM - PRINT ARRAY T
0170 PRINT "TABLE","A = ";S[1],
0180 PRINT "B = ";S[2],"C = ";S[3]
0190 PRINT
0200 FOR I=1 TO 4
0210   PRINT I,
0220   FOR J=1 TO 3
0230     PRINT T[I,J],
0240   NEXT J
0250   PRINT
0260 NEXT I
0270 END

*RUN)
TYPE A,B,C:
 ? 3)
 ? 15)
 ? 25)

TABLE          A =   3          B =   15          C =   25

 1               3                15                25
 2               6                30                50
 3               9                45                75
 4               12               60                100

END AT 0270
*
```

Exercise 7-4. Write a program that will input a two-dimensional array M, with R rows and C columns (Input values for R and C). Print Array M. Find and print the smallest element in each row and the largest element in each column.

Exercise 7-5. Write a program to input a 4 x 4 array and print out its transpose. The transpose of Array A is Array B in Figure 7-9.

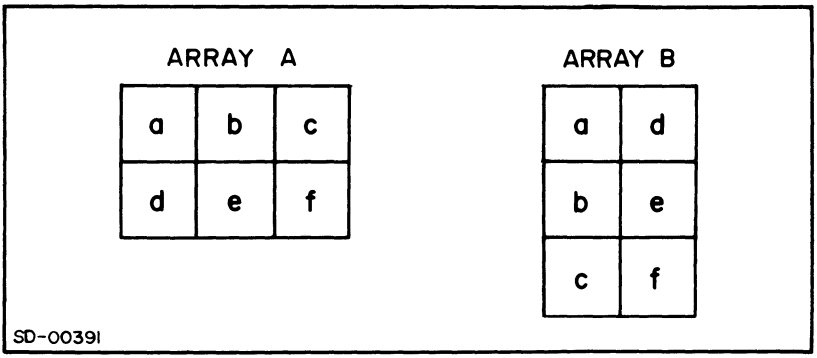


Figure 7-9. Array Transpose

END OF CHAPTER

CHAPTER 8

STRINGS

So far in BASIC you have been working only with numeric variables. Numbers are fine, but the alphabet is important too; eventually you will want to store both in memory. BASIC allows you to store any **alphanumeric** data as a **string**. A string is a sequence of characters enclosed in quotes which may include letters, digits, spaces and special characters.

STRING LITERALS

Whenever you've enclosed a message to BASIC in quotes (using the PRINT or INPUT statements), you've used a **string literal**. String literals have constant values; you can't change or rearrange them.

```
0060 PRINT "THIS IS A STRING LITERAL"
```

STRING VARIABLES

A **string variable** is a letter or a letter and a digit followed by a dollar sign (A$, B3$). It's value is a string. A string variable is the name of a memory location which will contain a string rather than a number. You may use string variables within a program in the same manner as numeric variables. (See Figure 8-1.)

```
*LIST)
0010 INPUT "TYPE YOUR NAME: ",N$
0020 PRINT N$;", YOU TYPE WELL."
0030 END

*RUN)
TYPE YOUR NAME: JOAN)
JOAN, YOU TYPE WELL.

END AT 0030
*
```

Figure 8-1. A Short String.

Lines 10 and 20 use string literals. $N defines the string variable. The input, JOAN, assigned a value to the string variable N$.

You can use string variables in place of numeric variables in any BASIC statement. Remember to enclose each string in quotation marks.

You may intermix string data and numeric data in a DATA list. However, the variables in the READ statement must match (numeric or string) the elements of the DATA list or BASIC will print an error message (Figure 8-2).

```
*LIST)
0010 DIM A$[25]
0020 READ A$
0030 PRINT A$
0040 DATA "THIS IS A STRING VARIABLE"
0050 END

*RUN)
THIS IS A STRING VARIABLE

END AT 0050
*
```

Figure 8-2. READ gets the String from the DATA List.

STRING SUBSCRIPTING

You may follow a string variable by a subscript but the concept is different from numberic subscripting. A string variable is stored with each character in a separate consecutively numbered memory location starting with 1. The DIM statement sets aside a row of locations in memory with as many locations as the maximum number of characters in the variable.

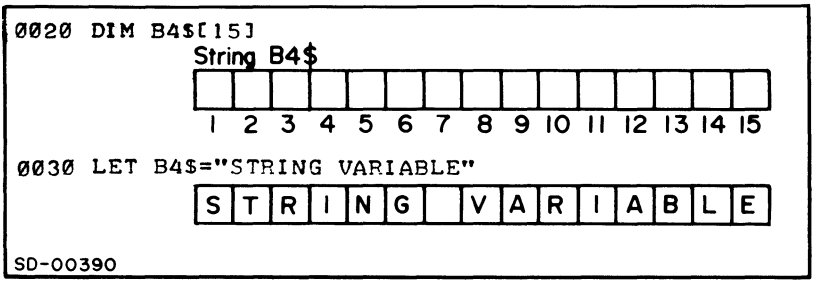


Figure 8-3. Programming Statements and Resulting Memory Locations

In Figure 8-3, line 20 assigns space for 15 characters to B4$. Line 30 fills the locations with the characters "STRING VARIABLE".

The DIM statement allows you to create string variables of any length, limited only by available memory. But BASIC will never print more characters than you specified in DIM; it will truncate your string if you haven't DIMensioned enough space (Figure 8-4). If you omit the DIM statement, your string variable will be truncated if it contains more than 10 characters.

```
*LIST)
0010 REM - STRING VARIABLE
0020 DIM A2$[15]
0030 LET A2$="PRINT A2$ IS THIRTY CHARACTERS"
0040 PRINT A2$
0050 END

*RUN)
PRINT A2$ IS TH

END AT 0050
*
```

Figure 8-4. A2$ is too long.

To reference the complete string, use just the name of the string variable. To reference part of a string (a **substring**), you can specify starting and stopping locations within the string. If you wish a substring to end with the last character in the string, specify the starting location. To reference one character of the string, give its character position as both start and stop locations. The program in Figure 8-5 demonstrates string subscripting.

```
*LIST)
0010 REM - STRING SUBSCRIPTING
0020 DIM B4$[15]
0030 LET B4$="STRING VARIABLE"
0040 PRINT "B4$       = ";B4$
0050 PRINT "B4$(1,6)  = ";B4$[1,6]
0060 PRINT "B4$(8)    = ";B4$[8]
0070 PRINT "B4$(5,5)  = ";B4$[5,5]
0080 END

*RUN)
B4$       = STRING VARIABLE
B4$(1,6)  = STRING
B4$(8)    = VARIABLE
B4$(5,5)  = N

END AT 0080
*
```

Figure 8-5. String Subscripting.

The program in Figure 8-6 demonstrates the use of numeric variables in place of digits in string subscripts.

```
*LIST)
0010 REM - STRING SUBSCRIPTING
0020 DIM A1$[20]
0030 LET A1$[2*1/2,5-2]="SUB"
0040 LET A=4
0050 LET B=10
0060 LET A1$[A,B]="GARBAGE"
0070 LET A1$[11,17]="EXAMPLE"
0080 LET A1$[A,B]="STRING "
0090 PRINT A1$
0100 END
*
```

Figure 8-6. Numeric Variables in String Subscripts.

Exercise 8-1. In Figure 8-6 what will BASIC print?

Exercise 8-2. Write a program to self-test the person running it. Print a vocabulary list with 10 words, and scramble definitions. Have the user match the definition to the word. Include instructions for taking the test, error and encouragement messages and a score at the end. Code a second try if the correct word is missed on the first one.

Exercise 8-3. Use strings to write a program to play an animal game. Read the name of an animal and have the player try to guess it. If his first guess is wrong, give him the first letter for a clue. If he still guesses wrong, give him the second letter. Continue until he guesses the animal.

END OF CHAPTER

APPENDIX A

PROBLEM ANSWERS

Exercise 3-1.

```
*LIST)
0010 REM - DIVIDE AND MULTIPLY
0020 PRINT 7/3
0030 PRINT 6*4
0040 END

*RUN)
 2.33333
 24

END AT 0040
*
```

Exercise 3-2.

0040 LET A+B=C is not a valid BASIC statement because A+B is not a valid variable name

Exercise 3-3.

```
*LIST)
0010 REM ** USES VARIABLES
0020 LET A=2
0030 LET B=3
0040 LET C=4
0050 LET D=A*B-C
0060 PRINT D
0070 END

*RUN)
 2

END AT 0070
*
```

Exercise 3-4.

```
*LIST)
0010 REM - ADD 2 AND 3
0030 LET B=3
0035 LET D=2
0040 LET C=B+D
0052 PRINT C
0060 END

*35 LET A=5)
*40 LET C=A)
*45 LET C=A+B)
*50 ;←PRINT A)
*51 PRINT B)
*RENUMBER)
*LIST)
0010 REM - ADD 2 AND 3
0020 LET B=3
0030 LET A=5
0040 LET C=A
0050 LET C=A+B
0060 PRINT A
0070 PRINT B
0080 PRINT C
0090 END

*RUN)
 5
 3
 8

END AT 0090
*
```

Exercise 3-5.

```
*LIST)
0010 REM - RETURNS +1
0020 LET A=-201.567
0030 LET A=INT(A)
0040 PRINT A
0050 LET A=ABS(A)
0060 PRINT A
0070 LET A=SGN(A)
0080 PRINT A
0090 END

*RUN)
-202
 202
 1

END AT 0090
*20 LET A=.321)
*RUN)
 0
 0
 0

END AT 0090
*20 LET A=42)
*70 LET A=-SGN(A))
*RUN)
 42
 42
-1

END AT 0090
*
```

INT(A) is not necessary in this program. ABS(A) and SGN(A) may be in either order. To return A=1 insert

```
0075 LET A=A-1
```

A = 0 will not return a value of 1

Exercise 4-1.

```
*LIST)
0010 REM - CALCULATE RECTANGLE AREA
0020 REM - INPUT LENGTH AND WIDTH
0030 INPUT X
0040 INPUT Y
0050 LET X=X*Y
0060 PRINT X
0070 END

*RUN)
 ? 4)
 ? 8)
 32

END AT 0070
*RUN)
 ? 75)
 ? 341)
 25575

END AT 0070
*
```

Exercise 4-2.

```
*LIST)
0010 REM SIMPLE INTEREST
0020 INPUT "TYPE PRINCIPAL: ",P
0030 INPUT "TYPE INTEREST RATE (DECIMAL): ",R
0040 INPUT "TYPE NUMBER OF YEARS: ",Y
0050 LET I=P*R*Y
0060 LET N=P+I
0070 LET R=R*100
0080 PRINT "$";P;" AT ";R;
0090 PRINT "% INTEREST FOR ";Y;" YEARS"
0100 PRINT "YIELDS $";I;" INTEREST."
0110 PRINT "NEW PRINCIPAL = $";N
0120 END

*RUN)
TYPE PRINCIPAL: 3500)
TYPE INTERST RATE (DECIMAL): .08)
TYPE NUMBER OF YEARS: 7)
$ 3500  AT  8 % INTEREST FOR  7  YEARS
YIELDS $ 1960  INTEREST.
NEW PRINCIPAL = $ 5460

END AT 0120
*
```

Exercise 4-3.

```
*LIST)
0010 REMARKS. WAGE, SALARY
0020 READ N
0030 PRINT N;
0040 READ H,W
0050 LET A=H*W
0060 PRINT H;"HRS ";W;"/HR",A;"SALARY"
0070 DATA 123,40,3.75
0080 END

*RUN)
 123   40 HRS   3.75 /HR         150 SALARY

END AT 0080
*
```

Exercise 4-4.

```
*LIST)
0010 REM - EXERCISE 4.4
0020 REM - A,B,C ARE SIDES
0030 REM - A1,B1,C1 ARE ANGLES
0040 REM - P CONVERTS FROM DEGREES TO RADIANS
0050 LET B=4
0060 LET C=5
0070 LET P=3.14159/180
0080 LET B1=45*P
0090 LET C1=65*P
0100 LET A=B*COS(C1)+C*COS(B1)
0110 PRINT "SIDE A =";A
0120 END

*RUN)
SIDE A = 5.2260128

END AT 0120
*
```

Exercise 4-5.

```
*LIST)
0010 REM ##### PRINT ME
0020 PRINT A,B,C
0030 READ X,Y,Z
0040 PRINT X;
0050 PRINT Y;
0060 PRINT Z
0070 PRINT
0080 PRINT "PLAYING COMPUTER IS FUN"
0090 PRINT 7,
0100 PRINT "6,5",4,
0110 PRINT
0120 PRINT "I CAN COUNT BACKWARDS",
0130 PRINT 3;2;1
0140 PRINT "THE END"
0150 DATA 5,7,9
0160 END

*RUN)
 0             0             0
 5  7  9

PLAYING COMPUTER IS FUN
 7            6,5           4
I CAN COUNT BACKWARDS        3  2  1
THE END

END AT 0160
*
```

Exercise 5-1.

```
*LIST)
0010 REM - FOLLOWING GOTO
0020 PRINT "DEMO OF GOTO"
0030 GOTO 0080
0040 PRINT "HAVE A GOOD DAY!"
0050 GOTO 0130
0060 PRINT "REALLY SKIPS AROUND, ";
0070 GOTO 0110
0080 PRINT
0090 PRINT "THIS PROGRAM ";
0100 GOTO 0060
0110 PRINT "DOESN'T IT?"
0120 GOTO 0040
0130 END

*RUN)
DEMO OF GOTO

THIS PROGRAM REALLY SKIPS AROUND, DOESN'T IT?
HAVE A GOOD DAY!

END AT 0130
*
```

Exercise 5-2.

```
*LIST)
0010 REMARKS. WAGE,SALARY
0020 PRINT "EMP #","HOURS","REGULAR PAY"
0030 READ N,H,W
0040 IF H>40 THEN GOTO 0120
0050 LET S=H*W
0060 LET S1=S
0070 PRINT N,H,S1
0080 PRINT
0090 PRINT "   ","OVERTIME PAY","TOTAL SALARY"
0100 PRINT "   ",S2,S
0110 STOP
0120 LET S1=40*W
0130 LET O=H-40
0140 LET W=W*1.5
0150 LET S2=O*W
0160 LET S=S1+S2
0170 GOTO 0070
0180 DATA 2,40,2

*RUN)
EMP #          HOURS          REGULAR PAY
 2              40             80

               OVERTIME PAY   TOTAL SALARY
               0              80

STOP AT 0110
*180 DATA 3,48,3.25)
*RUN)
EMP #          HOURS          REGULAR PAY
 3              48             130

               OVERTIME PAY   TOTAL SALARY
               39             169

STOP AT 0110
*
```

Exercise 5-3.

```
*LIST)
0010 REM  NUMBER RELATIONS
0020 INPUT X,Y
0030 IF X<Y THEN GOTO 0070
0040 IF X>Y THEN GOTO 0090
0050 PRINT X;" IS EQUAL TO ";Y
0060 GOTO 0100
0070 PRINT X;" IS LESS THAN ";Y
0080 GOTO 0100
0090 PRINT X;" IS GREATER THAN ";Y
0100 END

*RUN)
 ? 5,3)
 5  IS GREATER THAN  3

END AT 0100
*
```

Exercise 5-4.

```
*LIST)
0010 REM - FIND N FACTORIAL
0020 INPUT N
0030 PRINT N;
0040 LET X=1
0050 LET X=X*N
0060 LET N=N-1
0070 IF N THEN GOTO 0100
0080 PRINT " = ";X
0090 STOP
0100 PRINT "*";N;
0110 GOTO 0050

*RUN)
 ? 6)
 6 * 5 * 4 * 3 * 2 * 1  =  720

STOP AT 0090
*
```

Exercise 5-5.

```
*LIST)
0010 REM - PRINT HIGHEST NUMBER INPUT
0020 INPUT "TYPE NUMBER IN LIST: ",N
0030 PRINT "TYPE NUMBERS (FOLLOWED BY CR):"
0040 LET L=5.4E-79
0050 IF N THEN GOTO 0080
0060 PRINT "THE HIGHEST NUMBER IS ";L
0070 STOP
0080 INPUT X
0090 IF X>L THEN LET L=X
0100 LET N=N-1
0110 GOTO 0050

*RUN)
TYPE NUMBER IN LIST: 4)
TYPE NUMBERS (FOLLOWED BY CR):
 ? 47)
 ? 0)
 ? -18)
 ? 534)
THE HIGHEST NUMBER IS  534

STOP AT 0070
*
```

Exercise 5-6.

```
*LIST)
0010 REM - PRINT FRACTIONAL PART OF NUMBER
0020 REM - WITH PROPER SIGN
0030 REM - END ON 1000
0040 INPUT A
0050 IF A<>1000 THEN GOTO 0070
0060 STOP
0070 LET D=A-INT(A)
0080 IF A<0 THEN IF D<>0 THEN LET D=D-1
0090 PRINT D
0100 GOTO 0040
0110 END

*RUN)
 ? 4.32)
 .32
 ? -5.64)
-.64
 ? 0)
 0
 ? 1000)

STOP AT 0060
*
```

Exercise 5-7.

```
*LIST)
0010 REM - GUESS RANDOM NUMBER
0020 RANDOMIZE
0030 PRINT "I WILL PICK A NUMBER"
0040 PRINT "BETWEEN 1 AND 100"
0050 PRINT "AND YOU TRY TO GUESS IT."
0060 PRINT "YOU SHOULD BE ABLE TO GUESS IT"
0070 PRINT "IN SEVEN TRIES"
0080 PRINT
0090 REM - GET NUMBERS
0100 LET N=INT(RND(0)*100)
0110 INPUT "WHAT IS YOUR GUESS? ",A
0120 REM - C=COUNT OF TRIES
0130 LET C=C+1
0140 REM - IF HIGH OR LOW, PRINT MESSAGE
0150 IF A<N THEN GOTO 0330
0160 IF A>N THEN GOTO 0350
0170 PRINT
0180 REM - CORRECT GUESS
0190 PRINT "YOU GUESSED IT!!"
0200 IF C<2 THEN GOTO 0230
0210 PRINT "YOU TOOK ";C;" TRIES."
0220 GOTO 0260
0230 PRINT "AND ON THE FIRST TRY!!!"
0240 PRINT "SUPER FANTASTIC!!!"
0250 REM - PLAY AGAIN ROUTINE
0260 PRINT "DO YOU WANT TO PLAY AGAIN?"
0270 INPUT "       (TYPE 0=NO, 1=YES):",Z
0280 PRINT
0290 LET C=0
0300 IF Z THEN GOTO 0100
0310 STOP
0320 REM - HIGH AND LOW MESSAGES
0330 PRINT A;" IS LOWER THAN MY NUMBER."
0340 GOTO 0370
0350 PRINT A;" IS HIGHER THAN MY NUMBER."
0360 REM - CHECK NUMBER OF TRIES
0370 IF C<7 THEN GOTO 0110
0380 PRINT
0390 PRINT "I WIN - YOU HAVE HAD 7 GUESSES."
0400 REM - GUESS AFTER 7 TRIES?
0410 PRINT "DO YOU WANT TO KEEP GUESSING? "
0420 INPUT "          (TYPE 0=NO, 1=YES): ",Z
0430 PRINT
0440 IF Z THEN GOTO 0110
0450 PRINT "MY NUMBER IS ";N
0460 GOTO 0260
```

Exercise 5-7 (Cont.).

```
*RUN)
I WILL PICK A NUMBER
BETWEEN 1 AND 100
AND YOU TRY TO GUESS IT.
YOU SHOULD BE ABLE TO GUESS IT
IN SEVEN TRIES

WHAT IS YOUR GUESS? 45)
 45  IS HIGHER THAN MY NUMBER.
WHAT IS YOUR GUESS? 20)
 20  IS LOWER THAN MY NUMBER.
WHAT IS YOUR GUESS? 32)
 32  IS LOWER THAN MY NUMBER.
WHAT IS YOUR GUESS? 41)
 41  IS HIGHER THAN MY NUMBER.
WHAT IS YOUR GUESS? 47)
 47  IS HIGHER THAN MY NUMBER.
WHAT IS YOUR GUESS? 37)
 37  IS LOWER THAN MY NUMBER.
WHAT IS YOUR GUESS? 39)
 39  IS LOWER THAN MY NUMBER.

I WIN - YOU HAVE HAD 7 GUESSES.
DO YOU WANT TO KEEP GUESSING?
          (TYPE 0=NO, 1=YES): 1)

WHAT IS YOUR GUESS? 40)

YOU GUESSED IT!!
YOU TOOK  8  TRIES.
DO YOU WANT TO PLAY AGAIN?
      (TYPE 0=NO, 1=YES):0)


STOP AT 0310
*
```

Exercise 5-8.

```
*LIST)
0010 REM - SUBROUTINE ENDS IF NUMBER
0020 REM -   NOT BETWEEN 0 AND 100
0030 REM ** PRINTS 100-A
0040 READ A
0050 GOSUB 0070
0060 GOTO 0040
0070 REM ** SUBROUTINE STARTS HERE
0080 IF A<0 THEN STOP
0090 IF A>100 THEN STOP
0100 PRINT "100-";A;"EQUALS";100-A
0110 RETURN
0120 DATA 36,4,12,145,72,-1
0130 END

*RUN)
100- 36 EQUALS 64
100- 4 EQUALS 96
100- 12 EQUALS 88

STOP AT 0090
*
```

Exercise 5-9.

```
*LIST)
0010 REM - PAIR 0' DICE, THE DIVINE GAME
0020 REM - INSTRUCTIONS
0030 PRINT "DO YOU WANT INSTRUCTIONS?"
0040 INPUT "   (TYPE 1=NO, 0=YES): ",T
0050 IF T THEN GOTO 0200
0060 PRINT
0070 PRINT "THE COMPUTER WILL THROW THE DICE"
0080 PRINT "AND TELL YOU THE THROW AND"
0090 PRINT "YOUR POINT, THE TOTAL OF THE DICE"
0100 PRINT "THROWN.  ON THE FIRST THROW, YOU"
0110 PRINT "WIN WITH A POINT OF 7 AND THE"
0120 PRINT "COMPUTER WINS WITH A POINT OF 12."
0130 PRINT "YOU MAY CONTINUE TO THROW TO TRY"
0140 PRINT "TO MATCH YOUR POINT BUT A 7 WHILE"
0150 PRINT "TRYING FOR A POINT IS A WIN FOR"
0160 PRINT "THE COMPUTER."
0170 PRINT
0180 REM - MAIN GAME
0190 RANDOMIZE
0200 PRINT
0210 GOSUB 0470
0220 IF P=7 THEN GOTO 0330
0230 IF P=12 THEN GOTO 0380
0240 LET P1=P
0250 PRINT "TRY FOR POINT?"
0260 INPUT "    (TYPE 1=NO, 0=YES):",T
0270 IF T THEN GOTO 0380
0280 PRINT "YOU ARE TRYING FOR POINT ";P1
0290 GOSUB 0470
0300 IF P=P1 THEN GOTO 0330
0310 IF P=7 THEN GOTO 0380
0320 GOTO 0280
0330 REM - YOU WIN ROUTINE
0340 PRINT "YOU WIN!!"
0350 LET S1=S1+1
0360 PRINT "YOU: ";S1,"COMPUTER: ";S2
0370 GOTO 0420
0380 REM - COMPUTER WINS ROUTINE
0390 PRINT "COMPUTER WINS!!"
0400 LET S2=S2+1
0410 PRINT "YOU: ";S1,"COMPUTER: ";S2
0420 REM - PLAY AGAIN ROUTINE
0430 PRINT "DO YOU WANT TO PLAY AGAIN?"
0440 INPUT "    (TYPE 1=NO, 0=YES):",T
0450 IF T THEN GOTO 0660
0460 GOTO 0200
```

Exercise 5-9 (Cont.).

```
0470 REM - GET THROW AND POINT - SUBROUTINE
0480 REM - DUMMY NUMBER
0490 PRINT "ROLL THE DICE"
0500 INPUT "(TYPE A NUMBER BETWEEN 1 AND 100)",X
0510 IF X>=1 THEN IF X<=100 THEN GOTO 0540
0520 PRINT "I SAID BETWEEN 1 AND 100!!"
0530 GOTO 0490
0540 GOSUB 0630
0550 LET D1=N
0560 GOSUB 0630
0570 LET D2=N
0580 LET P=D1+D2
0590 PRINT "YOU THREW ";D1;" AND ";D2
0600 PRINT "YOUR POINT IS ";P
0610 PRINT
0620 RETURN
0630 REM - THROW DICE - SUBROUTINE
0640 LET N=INT(RND(0)*6)+1
0650 RETURN
0660 END

*RUN)
DO YOU WANT INSTRUCTIONS?
   (TYPE 1=NO, 0=YES): 0)

THE COMPUTER WILL THROW THE DICE
AND TELL YOU THE THROW AND
YOUR POINT, THE TOTAL OF THE DICE
THROWN.  ON THE FIRST THROW, YOU
WIN WITH A POINT OF 7 AND THE
COMPUTER WINS WITH A POINT OF 12.
YOU MAY CONTINUE TO THROW TO TRY
TO MATCH YOUR POINT BUT A 7 WHILE
TRYING FOR A POINT IS A WIN FOR
THE COMPUTER.
```

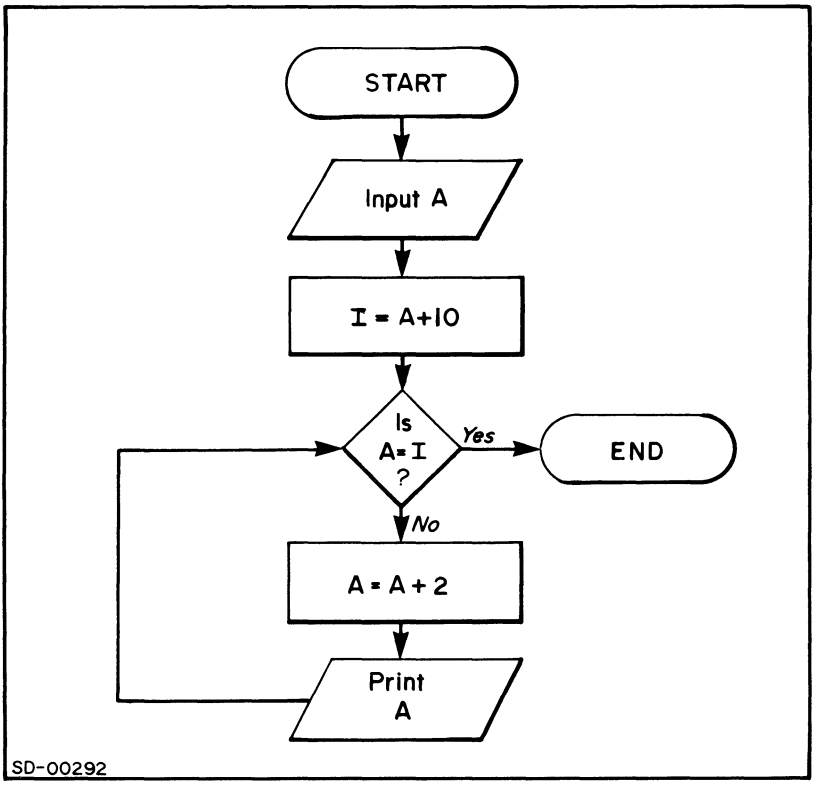
Exercise 5-9 (cont.).

```
ROLL THE DICE
(TYPE A NUMBER BETWEEN 1 AND 100)33)
YOU THREW  3  AND  2
YOUR POINT IS  5

TRY FOR POINT?
    (TYPE 1=NO, 0=YES):0)
YOU ARE TRYING FOR POINT  5
ROLL THE DICE
(TYPE A NUMBER BETWEEN 1 AND 100)25)
YOU THREW  5  AND  5
YOUR POINT IS  10

YOU ARE TRYING FOR POINT  5
ROLL THE DICE
(TYPE A NUMBER BETWEEN 1 AND 100)112)
I SAID BETWEEN 1 AND 100!!
ROLL THE DICE
(TYPE A NUMBER BETWEEN 1 AND 100)78)
YOU THREW  1  AND  6
YOUR POINT IS  7

COMPUTER WINS!!
YOU:  0        COMPUTER:  1
DO YOU WANT TO PLAY AGAIN?
    (TYPE 1=NO, 0=YES):1)

END AT 0660
*
```

Exercise 6-1.

I does not have to be incremented by 1. You can add 2 each time and test for I >10. You can start I at 5 and subtract 1 testing for I ≠ 0.

Another method of testing would be to add 10 (2*5) to A and test for that value.



Exercise 6-2.

```
*LIST)
0010 REM - PRINTS THE INTEGERS AND THEIR SQUARES
0020 REM - IF THE SQUARES ARE ODD
0030 PRINT "INTEGER","INTEGER SQUARED"
0040 FOR I=1 TO 25
0050   LET A=I*I
0060   LET A1=A/2
0070   LET A2=INT(A1)
0080   IF A1=A2 THEN GOTO 0100
0090   PRINT I,A
0100 NEXT I
0110 END

*RUN)
INTEGER        INTEGER SQUARED
 1              1
 3              9
 5              25
 7              49
 9              81
 11             121
 13             169
 15             225
 17             289
 19             361
 21             441
 23             529
 25             625

END AT 0110
*
```

Exercise 6-2A.

```
*LIST)
0010 REM - PRINTS THE INTEGERS AND THEIR SQUARES
0020 REM - IF THE INTEGER IN TENS PLACE IS ODD
0030 PRINT "NUMBER","SQUARED","TENS PLACE"
0040 FOR I=1 TO 50
0050   LET A=I*I
0060   LET A1=INT(A/10)
0070   LET A2=INT(A/100)
0080   LET N=A1-(A2*10)
0090   LET N1=N/2
0100   LET N2=INT(N1)
0110   IF N1=N2 THEN GOTO 0130
0120   PRINT I,A,N
0130 NEXT I
0140 END

*RUN)
NUMBER          SQUARED         TENS PLACE
 4               16              1
 6               36              3
 14              196             9
 16              256             5
 24              576             7
 26              676             7
 34              1156            5
 36              1296            9
 44              1936            3
 46              2116            1

END AT 0140
*
```

Exercise 6-3.

```
*LIST)
0010 REM - LIST THE FACTORS OF N
0020 INPUT N
0030 PRINT "  N","FACTORS OF N"
0040 PRINT N,
0050 FOR I=1 TO N
0060    LET N1=N/I
0070    IF INT(N1)<>N1 THEN GOTO 0090
0080    PRINT I;
0090 NEXT I
0100 END

*RUN)
 ? 125)
  N             FACTORS OF N
 125             1   5   25   125
END AT 0100
*
```

Exercise 6-4.

```
*LIST)
0010 REM - BALANCE CHECKBOOK
0020 INPUT "TYPE LAST BALANCE: ",A
0030 PRINT
0040 INPUT "TYPE NUMBER OF CHECKS WRITTEN: ",B
0050 GOSUB 0180
0060 LET A=A-X
0065 LET X=0
0070 PRINT
0080 INPUT "TYPE AMOUNT OF SERVICE CHARGE:  $",C
0090 LET A=A-C
0100 PRINT
0110 INPUT "TYPE NUMBER OF DEPOSITS: ",B
0120 GOSUB 0180
0130 LET A=A+X
0140 PRINT
0150 IF SGN(A)=-1 THEN PRINT "CHECKBOOK OVERDRAWN"
0160 PRINT "CURRENT BALANCE IS ";A
0170 STOP
0180 REM - SUBROUTINE
0190 FOR I=1 TO B
0200   INPUT "      AMOUNT?  $",Y
0210   LET X=X+Y
0220 NEXT I
0230 RETURN
0240 END

*RUN)
TYPE LAST BALANCE: 200)

TYPE NUMBER OF CHECKS WRITTEN: 3)
     AMOUNT?  $25.66)
     AMOUNT?  $328)
     AMOUNT?  $42.80)

TYPE AMOUNT OF SERVICE CHARGE:  $.15)

TYPE NUMBER OF DEPOSITS: 2)
     AMOUNT?  $100)
     AMOUNT?  $125)

CURRENT BALANCE IS  28.39

STOP AT 0170
*
```

Exercise 6-5.

```
*LIST)
0010 REM !!!!!! SQUARE ROOT SQUARED !!!!!!
0020 PRINT "NUMBER","SQUARE ROOT",
0030 PRINT "SQUARE ROOT SQUARED"
0040 FOR I=1 TO 10
0050   LET A=SQR(I)
0060   LET B=A*A
0070   PRINT I,A,B
0080 NEXT I
0090 END

*RUN)
NUMBER          SQUARE ROOT    SQUARE ROOT SQUARED
 1               1              1
 2               1.41421        2
 3               1.73205        3
 4               2              4
 5               2.23607        5
 6               2.44949        6
 7               2.64575        7
 8               2.82843        8
 9               3              9
 10              3.16228        10

END AT 0090
*
```

Exercise 6-6.

```
*LIST)
0010 REM ** COMPARES PRODUCT OF NUMBERS
0020 REM ** WITH EXPONENT OF SUM OF THEIR LOGARITHM
0030 INPUT "TYPE 2 NUMBERS: ",X,Y
0040 LET Z=X*Y
0050 LET X1=LOG(X)
0060 LET Y1=LOG(Y)
0070 LET Z1=EXP(X1+Y1)
0080 PRINT X;"=X",Y;"=Y",Z;"=X*Y"
0100 PRINT Z1;"=EXP(LOG(X)+LOG(Y))"
0110 END

*RUN)
TYPE 2 NUMBERS: 62,21)
 62 =X          21 =Y          1302 =X*Y
 1302 =EXP(LOG(X)+LOG(Y))

END AT 0110
*
```

Exercise 6-7.

```
*LIST)
0010 REM - MULTIPLICATION TABLES UP TO 5 TIMES 10
0020 FOR I=1 TO 5
0030   PRINT
0040   PRINT "MULTIPLICATION TABLE FOR ";I;":"
0050   FOR J=1 TO 10
0060     PRINT I*J;
0070   NEXT J
0080   PRINT
0090 NEXT I
0100 END

*RUN)

MULTIPLICATION TABLE FOR  1 :
 1  2  3  4  5  6  7  8  9  10

MULTIPLICATION TABLE FOR  2 :
 2  4  6  8  10  12  14  16  18  20

MULTIPLICATION TABLE FOR  3 :
 3  6  9  12  15  18  21  24  27  30

MULTIPLICATION TABLE FOR  4 :
 4  8  12  16  20  24  28  32  36  40

MULTIPLICATION TABLE FOR  5 :
 5  10  15  20  25  30  35  40  45  50

END AT 0100
*
```

Exercise 6-8.

```
*LIST)
0010 REM - LIST THE FACTORS OF N
0020 PRINT "  N","FACTORS OF N"
0030 FOR N=20 TO 30
0040   PRINT N,
0050   FOR I=1 TO N
0060     LET N1=N/I
0070     IF INT(N1)<>N1 THEN GOTO 0090
0080     PRINT I;
0090   NEXT I
0100   PRINT
0110 NEXT N
0120 END

*RUN)
  N             FACTORS OF N
 20              1  2  4  5  10  20
 21              1  3  7  21
 22              1  2  11  22
 23              1  23
 24              1  2  3  4  6  8  12  24
 25              1  5  25
 26              1  2  13  26
 27              1  3  9  27
 28              1  2  4  7  14  28
 29              1  29
 30              1  2  3  5  6  10  15  30

END AT 0120
*
```

Exercise 7-1.

```
*LIST)
0010 REM - FIND SQUARE ROOT OF ELEMENTS IN ARRAY
0020 REM - PRINT ARRAY ELEMENT THEN SQUARE ROOT
0030 PRINT "TYPE ARRAY ELEMENTS (FOLLOWED BY CR):"
0040 DIM T[15]
0050 DIM S[15]
0060 FOR I=1 TO 15
0070   INPUT T[I]
0080   LET S[I]=SQR(T[I])
0090 NEXT I
0100 PRINT
0110 PRINT "ELEMENT","SQUARE ROOT"
0120 FOR I=1 TO 15
0130   PRINT T[I],S[I]
0140 NEXT I
0150 END

*RUN)
TYPE ARRAY ELEMENTS (FOLLOWED BY CR):
 ? 5)
 ? 16)
 ? 28)
 ? 35)
 ? 43)
 ? 59)
 ? 61)
 ? 74)
 ? 82)
 ? 90)
 ? 105)
 ? 117)
 ? 234)
 ? 356)
 ? 470)
```

Exercise 7-1 (Cont.).

```
ELEMENT         SQUARE ROOT
 5               2.23607
 16              4
 28              5.2915
 35              5.91608
 43              6.55744
 59              7.68115
 61              7.81025
 74              8.60233
 82              9.05539
 90              9.48683
 105             10.247
 117             10.8167
 234             15.2971
 356             18.868
 470             21.6795

END AT 0150
*
```

Exercise 7-2.

```
*LIST)
0010 REM - PRINTS ARRAY WITH 10 ELEMENTS
0020 REM - FINDS HIGHEST AND LOWEST VALUES
0030 LET H=5.7E-79
0040 LET L=7.2E+75
0050 PRINT "TYPE 10 NUMBERS (FOLLOWED BY CR):"
0060 FOR I=1 TO 10
0070   INPUT T[I]
0080   IF T[I]<H THEN GOTO 0110
0090   LET H=T[I]
0100   LET H1=I
0110   IF T[I]>L THEN GOTO 0140
0120   LET L=T[I]
0130   LET L1=I
0140 NEXT I
0150 REM - PRINT ARRAY
0160 PRINT
0170 PRINT "ARRAY"
0180 PRINT
0190 FOR I=1 TO 10
0200   PRINT T[I];
0210   IF I=H1 THEN PRINT "      HIGHEST ELEMENT",
0220   IF I=L1 THEN PRINT "      LOWEST  ELEMENT",
0230   PRINT
0240 NEXT I
0250 END
```

Exercise 7-2 (Cont.).

```
*RUN)
TYPE 10 NUMBERS (FOLLOWED BY CR):
 ? 46)
 ? -321)
 ? 621)
 ? 0)
 ? 55)
 ? 738)
 ? 4E+10)
 ? 3)
 ? 72)
 ? 987654321)

ARRAY

 46
-321        LOWEST  ELEMENT
 621
 0
 55
 738
 4E+10        HIGHEST ELEMENT
 3
 72
 9.87654E+08

END AT 0250
*
```

Exercise 7-3.

```
*LIST)
0010 REM - PRINTS ARRAY IN ASCENDING ORDER
0020 INPUT "ARRAY SIZE? ",A
0030 IF A<=100 THEN GOTO 0060
0040 PRINT "ARRAY SIZE TOO BIG"
0050 GOTO 0020
0060 DIM T[A]
0070 PRINT "TYPE NUMBERS IN ARRAY:"
0080 FOR I=1 TO A
0090   INPUT T[I]
0100   FOR J=1 TO I
0110     IF T[I]>T[J] THEN GOTO 0150
0120     LET S=T[J]
0130     LET T[J]=T[I]
0140     LET T[I]=S
0150   NEXT J
0160 NEXT I
0170 PRINT
0180 PRINT
0190 FOR I=1 TO A
0200   PRINT T[I]
0210 NEXT I
0220 END

*RUN)
ARRAY SIZE? 212)
ARRAY SIZE TOO BIG
ARRAY SIZE? 7)
TYPE NUMBERS IN ARRAY:
 ? 5)
 ? 0)
 ? 345)
 ? -76)
 ? 5E+7)
 ? 13E=4\ ? 13E-4)
 ? 5432)


-76
 0
 .0013
 5
 345
 5432
 5E+07

END AT 0220
*
```

Exercise 7-4.

```
*LIST)
0010 REM ** READS TABLE WITH R ROWS AND C COLUMNS
0020 REM ** FINDS LOWEST ELEMENT IN EACH ROW
0030 REM **       HIGHEST ELEMENT IN EACH COLUMN
0040 INPUT "NUMBER OF ROWS? ",R
0050 INPUT "NUMBER OF COLUMNS? ",C
0060 DIM M[R,C]
0070 REM - FILL TABLE
0080 FOR I=1 TO C
0090   PRINT "TYPE IN COLUMN ";I
0100   FOR J=1 TO R
0110     INPUT X
0120     LET M[J,I]=X
0130   NEXT J
0140 NEXT I
0150 REM - PRINT M
0160 PRINT
0170 FOR I=1 TO R
0180   FOR J=1 TO C
0190     PRINT M[I,J],
0200   NEXT J
0210   PRINT
0220 NEXT I
0230 PRINT
0240 REM ** FIND LOWEST
0250 PRINT "LOWEST ROW ELEMENTS ARE:"
0260 FOR I=1 TO R
0270   LET H=7.2E+75
0280   FOR J=1 TO C
0290     IF H>M[I,J] THEN LET H=M[I,J]
0300   NEXT J
0310   PRINT H
0320 NEXT I
0330 REM ** FIND HIGHEST
0340 PRINT
0350 PRINT "HIGHEST COLUMN ELEMENTS ARE:"
0360 FOR I=1 TO C
0370   LET H=5.4E-79
0380   FOR J=1 TO R
0390     IF H<M[J,I] THEN LET H=M[J,I]
0400   NEXT J
0410   PRINT H;
0420 NEXT I
0430 PRINT
0440 END
```

Exercise 7-4 (Cont.).

```
*RUN)
NUMBER OF ROWS? 4)
NUMBER OF COLUMNS? 3)
TYPE IN COLUMN  1
 ? 7)
 ? 4)
 ? 1)
 ? 18)
TYPE IN COLUMN  2
 ? 6)
 ? 12)
 ? 0)
 ? 14)
TYPE IN COLUMN  3
 ? 2)
 ? 7)
 ? 10)
 ? 8)

 7              6              2
 4              12             7
 1              0              10
 18             14             8

LOWEST ROW ELEMENTS ARE:
 2
 4
 0
 8

HIGHEST COLUMN ELEMENTS ARE:
 18  14  10

END AT 0440
*
```

Exercise 7-5.

```
*LIST)
0010 REM ** INPUT MATRIX AND PRINT ITS TRANSPOSE
0020 DIM T[4,4]
0030 FOR I=1 TO 4
0040   PRINT "TYPE IN COLUMN ";I
0050   FOR J=1 TO 4
0060     INPUT T[I,J]
0070   NEXT J
0080 NEXT I
0090 PRINT
0100 PRINT "ORIGINAL MATRIX"
0110 FOR I=1 TO 4
0120   FOR J=1 TO 4
0130     PRINT T[J,I];
0140   NEXT J
0150   PRINT
0160 NEXT I
0170 PRINT
0180 PRINT "TRANSPOSED MATRIX"
0190 FOR I=1 TO 4
0200   FOR J=1 TO 4
0210     PRINT T[I,J];
0220   NEXT J
0230   PRINT
0240 NEXT I
0250 END
```

Exercise 7-5 (Cont.).

```
*RUN)
TYPE IN COLUMN  1
 ? 1)
 ? 2)
 ? 3)
 ? 4)
TYPE IN COLUMN  2
 ? 5)
 ? 6)
 ? 7)
 ? 8)
TYPE IN COLUMN  3
 ? 9)
 ? 10)
 ? 11)
 ? 12)
TYPE IN COLUMN  4
 ? 13)
 ? 14)
 ? 15)
 ? 16)

ORIGINAL MATRIX
 1   5   9   13
 2   6   10  14
 3   7   11  15
 4   8   12  16

TRANSPOSED MATRIX
 1   2   3   4
 5   6   7   8
 9   10  11  12
 13  14  15  16

END AT 0250
*
```

Exercise 8-1.

```
*LIST)
0010 REM - STRING SUBSCRIPTING
0020 DIM A1$[20]
0030 LET A1$[2*1/2,5-2]="SUB"
0040 LET A=4
0050 LET B=10
0060 LET A1$[A,B]="GARBAGE"
0070 LET A1$[11,17]="EXAMPLE"
0080 LET A1$[A,B]="STRING "
0090 PRINT A1$
0100 END

*RUN)
SUBSTRING EXAMPLE

END AT 0100
*
```

Exercise 8-2.

```
*LIST)
0010 RANDOMIZE
0020 REM - VOCABULARY SELF-TEST
0030 INPUT "DO YOU WANT INSTRUCTIONS? ",Z$
0040 IF Z$<>"YES" THEN GOTO 0230
0050 PRINT
0060 PRINT "THIS IS A VOCABULARY TEST INCLUDING"
0070 PRINT "INTRODUCTORY COMPUTER WORDS."
0080 PRINT "THE COMPUTER WILL PRINT A LIST OF"
0090 PRINT "WORDS FOLLOWED BY THEIR DEFINITIONS."
0100 PRINT "YOU MUST MATCH THE PROPER WORD WITH"
0110 PRINT "ITS DEFINITION."
0120 PRINT
0130 PRINT "IF YOU TYPE THE WRONG WORD, YOU "
0140 PRINT "WILL HAVE ANOTHER CHANCE TO PICK"
0150 PRINT "THE CORRECT ONE."
0160 PRINT
0170 PRINT "YOU WILL BE SCORED AS FOLLOWS:"
0180 PRINT "      CORRECT WORD FIRST  TRY ... 10"
0190 PRINT "      CORRECT WORD SECOND TRY ...  5"
0200 PRINT "                PERFECT SCORE ...100"
0210 PRINT
0220 PRINT "GOOD LUCK!"
0230 PRINT
0240 PRINT "HERE ARE YOUR VOCABULARY WORDS:"
0250 READ A$,B$,C$,D$,E$,F$,G$,H$,I$,J$
0260 GOSUB 0860
0270 PRINT
0280 PRINT "HERE ARE YOUR DEFINITIONS. TYPE THE"
0290 PRINT "CORRECT WORD AFTER EACH DEFINITION."
0300 PRINT
```

Exercise 8-2 (Cont.).

```
0310 LET X$=E$
0320 PRINT "BEGINNER'S ALL-PURPOSE SYMBOLIC"
0330 PRINT "INSTRUCTION CODE."
0340 GOSUB 0990
0350 LET X$=B$
0360 PRINT "A SYMBOLIC DIAGRAM OF THE LOGIC"
0370 PRINT "FLOW THROUGH A PROGRAM."
0380 GOSUB 0990
0390 LET X$=H$
0400 PRINT "CENTRAL PROCESSING UNIT --"
0410 PRINT "THE HEART OF THE COMPUTER."
0420 GOSUB 0990
0430 LET X$=G$
0440 PRINT "A BASIC WORD USED TO ASSIGN A VALUE"
0450 PRINT "TO A VARIABLE."
0460 GOSUB 0990
0470 LET X$=A$
0480 PRINT "A COMPUTER PRINTOUT OF A PROGRAM."
0490 GOSUB 0990
0500 PRINT
0510 PRINT "DO YOU WANT TO SEE THE LIST AGAIN?"
0520 INPUT Z$
0530 IF Z$="YES" THEN GOSUB 0860
0540 PRINT
0550 LET X$=I$
0560 PRINT "INFORMATION AND VALUES A PROGRAM"
0570 PRINT "USES TO PERFORM CALCULATIONS."
0580 GOSUB 0990
0590 LET X$=F$
0600 PRINT "A DATA NAME WHICH CAN CONTAIN"
0610 PRINT "DIFFERENT VALUES AT DIFFERENT"
0620 PRINT "TIMES IN A PROGRAM."
0630 GOSUB 0990
0640 LET X$=J$
0650 PRINT "A BASIC STATEMENT USED FOR"
0660 PRINT "INTERNAL DOCUMENTATION."
0670 GOSUB 0990
0680 LET X$=D$
0690 PRINT "A BASIC WORD WHICH IS EXECUTED"
0700 PRINT "AS SOON AS A CARRIAGE RETURN"
0710 PRINT "IS TYPED."
0720 GOSUB 0990
0730 LET X$=C$
0740 PRINT "A COMPUTERIZED TYPEWRITER USED"
0750 PRINT "TO INPUT DATA AND PROGRAM"
0760 PRINT "STATEMENTS TO A COMPUTER."
0770 GOSUB 0990
```

Exercise 8-2 (Cont.).

```
0780 PRINT
0790 PRINT "   YOUR SCORE: ";S;
0800 IF S=100 THEN PRINT " - SUPER!!";
0810 IF S>=80 THEN PRINT " YOU DID REALLY WELL!"
0820 DATA "LISTING","FLOW CHART","TERMINAL"
0830 DATA "COMMAND","BASIC","VARIABLE","LET"
0840 DATA "CPU","DATA","REM"
0850 STOP
0860 PRINT
0870 PRINT A$
0880 PRINT B$
0890 PRINT C$
0900 PRINT D$
0910 PRINT E$
0920 PRINT F$
0930 PRINT G$
0940 PRINT H$
0950 PRINT I$
0960 PRINT J$
0970 PRINT
0980 RETURN
0990 REM - CHECK ANSWER AND FIGURE SCORE
1000 INPUT Y$
1010 LET B=10
1020 LET A=INT(RND(0)*10)
1030 IF Y$<>X$ THEN GOTO 1080
1040 IF A<3 THEN PRINT "****GREAT****"
1050 IF A=5 THEN PRINT "!! FANTASTIC !!"
1060 IF A>5 THEN PRINT "    GOOD!"
1070 GOTO 1160
1080 LET B=B-5
1090 IF B THEN GOTO 1130
1100 PRINT "THE CORRECT ANSWER IS ";X$
1110 PRINT "GO ON TO THE NEXT ONE. "
1120 GOTO 1160
1130 PRINT "WRONG ANSWER, PICK ANOTHER."
1140 INPUT Y$
1150 GOTO 1030
1160 LET S=S+B
1170 PRINT
1180 RETURN
1190 END
```

Exercise 8-2 (Cont.).

```
*RUN)
DO YOU WANT INSTRUCTIONS? NO

HERE ARE YOUR VOCABULARY WORDS:

LISTING
FLOW CHART
TERMINAL
COMMAND
BASIC
VARIABLE
LET
CPU
DATA
REM


HERE ARE YOUR DEFINITIONS. TYPE THE
CORRECT WORD AFTER EACH DEFINITION.

BEGINNER'S ALL-PURPOSE SYMBOLIC
INSTRUCTION CODE.
 ? BASIC)
    GOOD!

A SYMBOLIC DIAGRAM OF THE LOGIC
FLOW THROUGH A PROGRAM.
 ? FLOW CHART)
    GOOD!

CENTRAL PROCESSING UNIT --
THE HEART OF THE COMPUTER.
 ? CPU)
    GOOD!

A BASIC WORD USED TO ASSIGN A VALUE
TO A VARIABLE.
 ? PRINT)
WRONG ANSWER, PICK ANOTHER.
 ? LET)
    GOOD!

A COMPUTER PRINTOUT OF A PROGRAM.
 ? LISTING)
```

Exercise 8-2 (Cont.).

```
DO YOU WANT TO SEE THE LIST AGAIN?
 ? NO)

INFORMATION AND VALUES A PROGRAM
USES TO PERFORM CALCULATIONS.
 ? NUMBERS)
WRONG ANSWER, PICK ANOTHER.
 ? VARIABLE)
THE CORRECT ANSWER IS DATA
GO ON TO THE NEXT ONE.

A DATA NAME WHICH CAN CONTAIN
DIFFERENT VALUES AT DIFFERENT
TIMES IN A PROGRAM.
 ? VARIABLE)
!! FANTASTIC !!

A BASIC STATEMENT USED FOR
INTERNAL DOCUMENTATION.
 ? REM)
****GREAT****

A BASIC WORD WHICH IS EXECUTED
AS SOON AS A CARRIAGE RETURN
IS TYPED.
 ? COMMAND)

A COMPUTERIZED TYPEWRITER USED
TO INPUT DATA AND PROGRAM
STATEMENTS TO A COMPUTER.
 ? TERMINAL)


   YOUR SCORE:  85  YOU DID REALLY WELL!

STOP AT 0850
*
```

Exercise 8-3.

```
*LIST)
0010 REM ** GUESS THE ANIMAL GAME
0020 REM ** CHANGE DATA FOR DIFFERENT ANIMALS
0030 GOSUB 0250
0040 PRINT
0050 PRINT "I AM THINKING OF AN ANIMAL."
0060 INPUT "SEE IF YOU CAN GUESS IT: ",Z$
0070 GOSUB 0180
0080 PRINT "NO, THAT ISN'T IT."
0090 PRINT "IT STARTS WITH ";A$[1,1]
0100 LET A=1
0110 PRINT
0120 INPUT "GUESS AGAIN: ",Z$
0130 GOSUB 0180
0140 PRINT "STILL NOT RIGHT."
0150 LET A=A+1
0160 PRINT "THE NEXT LETTER IS ";A$[A,A]
0170 GOTO 0110
0180 REM - GUESS RIGHT?
0190 IF A$<>Z$ THEN RETURN
0200 PRINT
0210 PRINT "HEY - YOU GUESSED IT!!"
0220 GOSUB 0250
0230 INPUT "DO YOU WANT TO PLAY AGAIN? ",Y$
0240 IF Y$[1,1]="Y" THEN GOTO 0040
0250 REM - AT END?
0260 READ A$
0270 IF A$<>"END" THEN RETURN
0280 PRINT "I AM OUT OF ANIMALS"
0290 DATA "ELEPHANT","TURTLE","END"
0300 END
```

Exercise 8-3 (Cont.).

```
*RUN)

I AM THINKING OF AN ANIMAL.
SEE IF YOU CAN GUESS IT: DOG)
NO, THAT ISN'T IT.
IT STARTS WITH E

GUESS AGAIN: ELEPHANT )

HEY - YOU GUESSED IT!!
DO YOU WANT TO PLAY AGAIN? YES)

I AM THINKING OF AN ANIMAL.
SEE IF YOU CAN GUESS IT: CAT)
NO, THAT ISN'T IT.
IT STARTS WITH T

GUESS AGAIN: TURKEY)
STILL NOT RIGHT.
THE NEXT LETTER IS U

GUESS AGAIN: TURTLE)

HEY - YOU GUESSED IT!!
I AM OUT OF ANIMALS

END AT 0300
*
```

END OF APPENDIX

APPENDIX B

ERROR MESSAGES

Errors are inevitable; everyone makes them. This appendix lists BASIC's error messages and describes possible causes.

Extended BASIC prints error messages as 2 digit codes, usually followed by a brief explanatory message. BASIC prints error messages when it cannot understand or is unable to perform some command or statement, often because you typed it improperly.

BASIC recognizes some errors during program input. Whenever you enter an incorrect statement or misuse a keyword, it will return an error message. If you are at a terminal, this message will refer to your last statement. If a card deck, paper tape reader, or magnetic tape drive entered the incorrect statement, BASIC will print the statement that caused the error message.

If the error occurs while your program is running, the error message will include the line number of the statement which caused it.

The following chart lists BASIC error messages, explains their meaning and shows the kind of error that caused them. Use this chart when you get an error message and need more information than the error text provides. This list doesn't include advanced BASIC error messages associated with BASIC features we haven't explained. For a complete error message list please see Data General's Extended BASIC User's Manual.

BASIC Error Messages

Code	Text	Meaning	Example(s)
00	FORMAT	unrecognizable statement format	LET A==2)
01	CHARACTER	illegal or unexpected character	PRINT #HI) NEW%%) 10&&REM)
02	SYNTAX	invalid argument type	10 DIM A(2) 20 IF SIN(A$)=0) LET 10=A)
03	READ/DATA TYPES	READ specifies different data type than DATA statement	20 READ A,B) 30 DATA 12, "HI") RUN)
04	SYSTEM	hardware or software malfunction	
05	STATEMENT NUMBER	statement number greater than 9999	10 GOTO 12345)
11	PARENTHESES	parentheses in an expression are not paired	LET A = ((B-C))
12	COMMAND	keyword unrecognizable, statement instead of command	FOR J = 1 to 5) LETTA=10) PRNIT "HELLO")
13	LINE NUMBER	attempt to delete or list an unknown line; attempt to transfer to an unknown line	100) 10 GOTO 100) RUN)

BASIC Error Messages (Continued)

Code	Text	Meaning	Example(s)
15	END OF DATA	not enough DATA arguments to satisfy READ	10 READ A,B,C 20 DATA 91,21 RUN
16	ARITHMETIC	value too large or too small to evaluate; or a division by 0	LET A = 1234E+76 PRINT 5/0
18	GOSUB NESTING	too many nested GOSUB'S	10 GOSUB 20 20 GOTO 10 RUN
19	RETURN - NO GOSUB	RETURN statement encountered without a corresponding GOSUB	NEW 10 RETURN RUN
20	FOR NESTING	too many nested FOR/NEXT loops	10 FOR A=1 to 3 20 FOR B=10 to 15 30 FOR C=5 to 7 ⋮ 100 FOR J=2 to 8 RUN
21	FOR - NO NEXT	unexecutable FOR - NEXT loop; FOR without a NEXT	10 FOR A = 1 TO 3 20 PRINT A 30 RUN
22	NEXT - NO FOR	NEXT statement without a corresponding FOR	NEW 10 NEXT I RUN
23	DATA OVERFLOW	not enough storage left for variables	10 DIM A(30000) RUN
28	DIM OVERFLOW	an array or string exceeds its initial dimensions	10 DIM A(2) 20 DIM A(5) RUN

BASIC Error Messages (Continued)

Code	Text	Meaning	Examples
31	SUBSCRIPT	subscript exceeds DIMension of array or string	10 DIM A(2) 20 PRINT A(3) RUN
46	INPUT	too many responses to INPUT	10 INPUT A RUN ? 1,2
53	RENUMBER	After a RENUMBER command, BASIC encountered a non-existant statement	NEW 10 GOTO 100 RENUMBER

END OF APPENDIX

APPENDIX C

PROGRAMMING

ON MARK-SENSE CARDS

You may write BASIC programs on Data General's Extended BASIC mark-sense programming cards for input to the mark sense card reader.

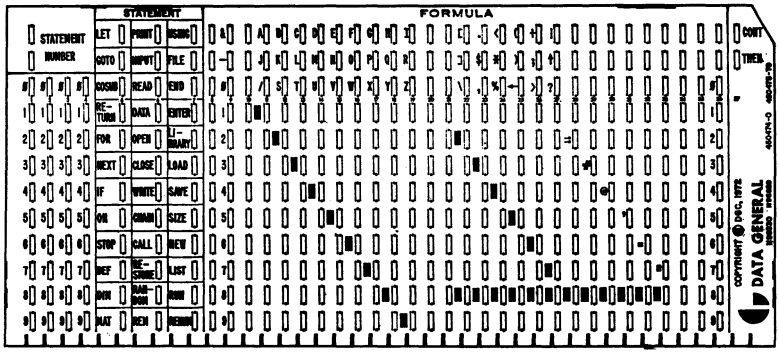


Figure C-1. Data General Extended BASIC Mark-sense Programming Card

You may mark a stack of cards to include an entire BASIC program, and input your stack to the card reader as a **batch** job. Your system manager will know about any special cards your system may require.

The mark-sense reader has an option which permits either markings or punches. With this option, you may punch mark-sense cards. Marked and punched cards may be intermixed in a deck; a single card may be both marked and punched. You must use a a No. 2 pencil to mark cards.

A Data General Extended BASIC mark-sense card has 37 columns, as shown in figure C-1. The first four columns assign statement numbers; the next three assign the BASIC statement keyword. A single BASIC statement or part of a statement may be written on each card.

The BASIC statement field of the mark-sense card is three columns which allow all possible combinations of statement keywords. Cards are marked in the appropriate column, for example; we have marked the statement 450 GOTO 200 in Figure C-2.

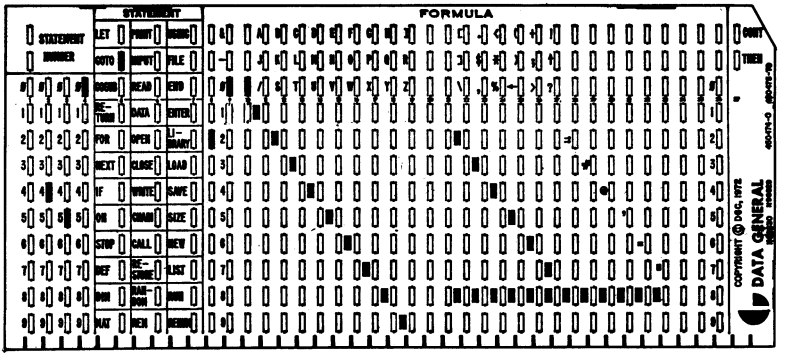


Figure C-2. 450 GOTO 200

The formula section of the card is 29 columns long, and 12 rows deep. You proceed from the left-most column to the right, for up to 29 characters; the CONT box on the far right allows you to continue your statement to the next card.

You must fill out the formula section of each card in Hollerith code. Each Data General mark-sense card contains a Hollerith code key (the black squares in the formula section), which indicates the lines that you must mark for each character. On all mark-sense cards numbers are marked directly in the appropriate row, without the key. Letters require two marks in a row, and special characters either two or three marks.

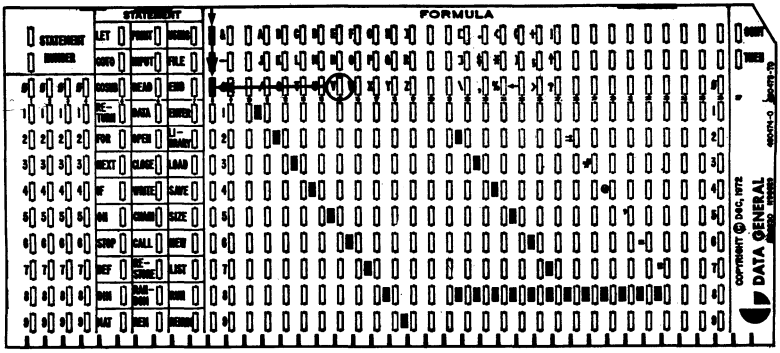


Figure C-3. Let's mark V.

On mark-sense cards, find the character column and the character you want to mark. We've indicated the first column with an arrow and the letter V as the character we will mark. Mark the rectangle at the intersection of the two arrows. (Figure C-3).

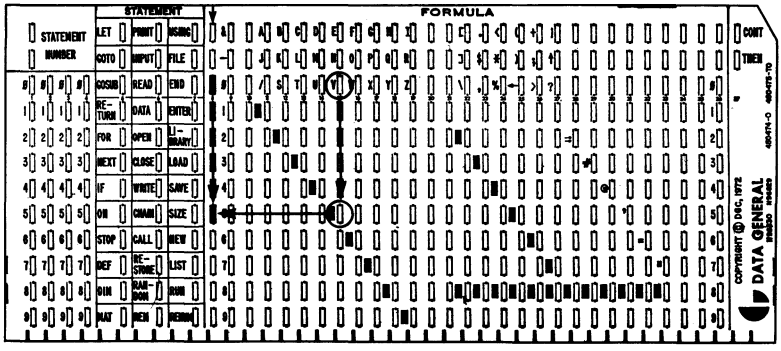


Figure C-4.

Find any boxes directly under your character. We have circled the box under the V. Some characters have more than one box below them. Again, mark the rectangle at the intersection of the arrows (Figure C-4).

Do not draw the arrows on the mark-sense cards as BASIC will try to interpret them. The first column contains the completed markings for the character V. (Figure C-5).

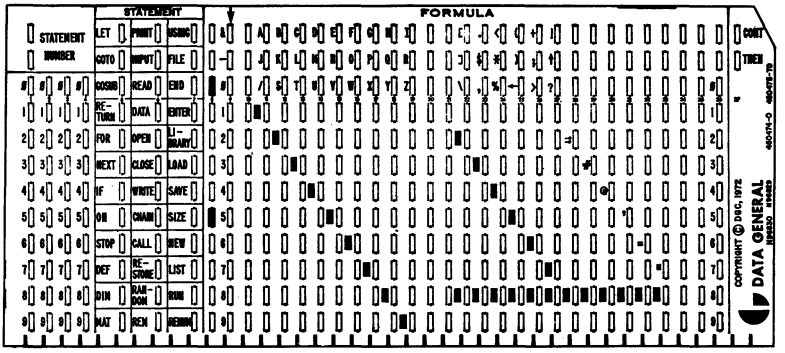


Figure C-5.

Move one column over and find your next character. Begin again.

You may be using cards without a key. If so, fill them out according to the Hollerith character set at the end of this appendix. The mark-sense card key and the Hollerith character set work exactly the same way; you may use whichever you find easier. If you use the Hollerith code set, the top horizontal line is number 12, the second from the top is number 11, and the other lines are numbered from 0 through 9. To indicate 4, put a mark on line 4; to indicate an asterisk (*), put marks on lines 11, 4 and 8; to indicate a number sign (#), put marks on lines 3 and 8.

On any card, you can continue a statement to the next card by marking the CONT box in the upper righthand corner of the first card. Continue the statement on the following card in the FORMULA section.

To write an IF statement, mark IF in the statement section, mark the test expression in the formula section, and mark the THEN box in the upper right-hand corner of the card. On the next card, begin continue in the formula section.

To further illustrate the use of mark-sense cards, we have coded 10 IF V$ = "CAT" THEN in Figure C-6.

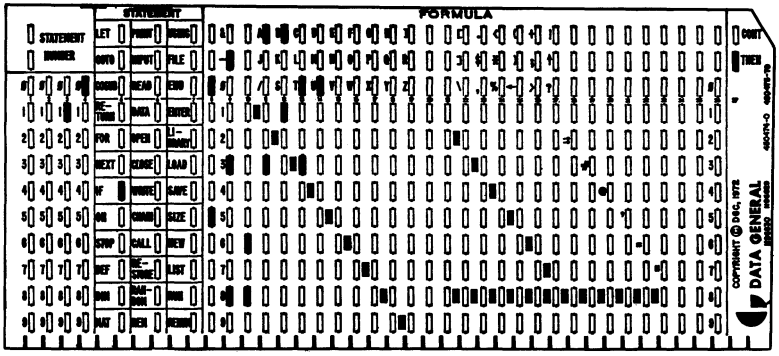


Figure C-6. 10 IF V$ = "CAT" THEN

HOLLERITH CHARACTER SET

Use this table on the formula section of mark-sense cards (see page C-2).

Character	Lines			Character	Lines		
0	0	-	-	J	11	1	-
1	1	-	-	K	11	2	-
2	2	-	-	L	11	3	-
3	3	-	-	M	11	4	-
4	4	-	-	N	11	5	-
5	5	-	-	O	11	6	-
6	6	-	-	P	11	7	-
7	7	-	-	Q	11	8	-
8	8	-	-	R	11	9	-
9	9	-	-	S	0	2	-
A	12	1	-	T	0	3	-
B	12	2	-	U	0	4	-
C	12	3	-	V	0	5	-
D	12	4	-	W	0	6	-
E	12	5	-	X	0	7	-
F	12	6	-	Y	0	8	-
G	12	7	-	Z	0	9	-
H	12	8	-	[	12	2	8
I	12	9	-	.	12	3	8

Character	Lines		
<	12	4	8
(	12	5	8
+	12	6	8
!	12	7	8
]	11	2	8
$	11	3	8
*	11	4	8
)	11	5	8
;	11	6	8
↑	11	7	8
\	0	2	8
, (comma)	0	3	8
%	0	4	8
←	0	5	8
>	0	6	8
?	0	7	8
:	2	8	-
#	3	8	-
@	4	8	-

Character	Lines		
'	5	8	-
=	6	8	-
"	7	8	-
&	12	-	-
- (minus)	11	-	-

END OF APPENDIX

INDEX/GLOSSARY