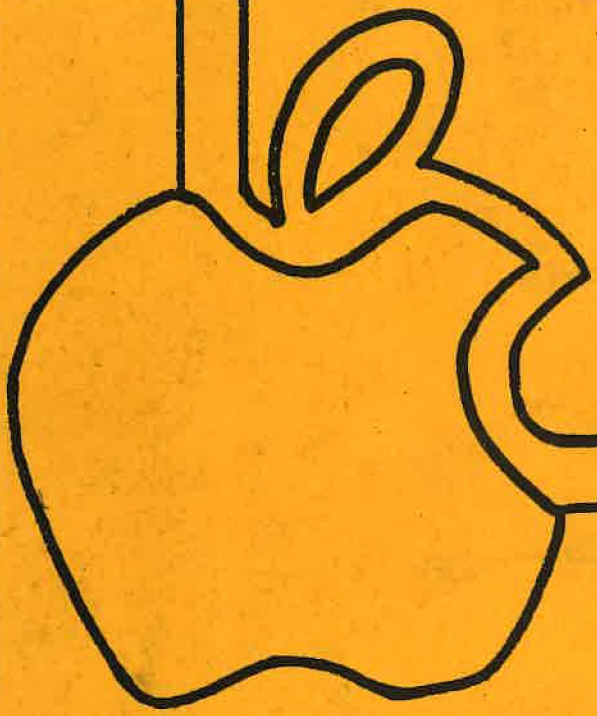
INTRODUCTION TO APPLESOFT™ BASIC

STUDENT TEXT



PRELIMINARY VERSION

FOR USE WITH

THE APPLE II™ MICROCOMPUTER

MINNESOTA EDUCATIONAL COMPUTING CONSORTIUM

TABLE OF CONTENTS

INTRODUCTION

This student manual introduces elementary and intermediate level topics of the Applesoft™ BASIC programming language for the APPLE II™ microcomputer. It is intended for use in a classroom lecture setting with assignments given to students. No prior programming knowledge is required on the part of the students using this manual. The manual might be used in a class solely devoted to computer programming or as a means of introducing programming concepts in a "computer awareness" course that also covers topics such as the history of computing and the impact of computers in society.

Since mastery of the concepts and skills covered in the manual is best achieved by active experience, it is highly recommended that examples and exercises be entered and executed on an APPLE II. The mechanics of using the APPLE II for entering and saving programs on diskettes is not, however, covered in this manual. For this information, the reader is referred to the MECC APPLE New User's Guide, available from MECC Publications (see page 126).

The time required to progress through the material in this manual will vary based on the amount of computer time available to students on an APPLE II. However, the student with no prior programming experience should cover the material in about six weeks, assuming five hours of classroom instruction each week.

Answers to the exercises are available in a separate booklet from MECC Publications.

ACKNOWLEDGEMENTS

This manual was written by John Arneson and Willis Jokela of the MECC Instructional Services staff, based on material from a similar MECC Timeshare programming booklet by Gary Schafer. Dick Quast of Morgan (MN) Public Schools helped in reviewing the document.

Chapter I

OUTPUT/INPUT

1 - 1 Getting Started

This manual is designed to help you get started writing programs in APPLESOFT BASIC on the APPLE II microcomputer. The word BASIC means Beginners All-Purpose Symbolic Instruction Code. It was developed at Dartmouth College to allow students to write programs in an easy-to-learn English-like language.

BASIC STATEMENT STRUCTURE

A BASIC statement is an instruction to the computer to take some action or perform a task. Each BASIC statement has the form:

LN KEYWORD PARAMETERS

where:

LN -- is a legal line number. These are integers Ø through 63999. They determine the order in which statements are executed.

KEYWORD -- is a special word telling the computer which operation is to be performed.

PARAMETERS -- provide additional information to the computer to complete the operation.

The KEYWORDS used in this text are:

PRINT	GET	VTAB
END	COLOR	HTAB
LET	GR	HGR
GO TO	TEXT	HCOLOR
IF THEN	HOME	HPLOT
INPUT	HLIN	
READ	VLIN	
DATA		
FOR		
NEXT		
DIM		

Notice that KEYWORDS have the appearance of English words. Learning to program begins with learning a set of rules for writing STATEMENTS using these KEYWORDS.

RULES FOR WRITING A BASIC STATEMENT

Every statement must have a legal line number.

No two statements can have the same line number.

Exact spelling of keywords must be used.

1 - 2 Printing Messages

PRINTING MESSAGES

To use the APPLE II microcomputer, we need a method of getting it to print information on its screen. The PRINT statement causes the computer to print information.

general form: LN PRINT "ANY MESSAGE OR STRING OF CHARACTERS"

example: 10Ø PRINT "PRINTING MESSAGES IS EASY AS A B C"

purpose: Messages are strings of characters enclosed in quote marks in a PRINT statement. The microcomputer will print them, exactly as they are typed.

ENDING A PROGRAM

Each program should contain a statement telling the computer where the program ends.

general form: LN END

example: 2ØØ END

purpose: Since statements are executed in line number order, the END statement must have the largest line number in the program. END signals the microcomputer that no more statements are to follow and directs it to terminate the program.

You are now ready to write computer programs. Study the following simple problems and then work the exercises.

sample 1: Write a program to print the message: MY COMPUTER LIKES ME

solution:

```
100  PRINT "MY COMPUTER LIKES ME"
200  END
```

sample 2: Write a program to print the city, state, and zip code where you live:

solution:

```
10  PRINT "MARSHALL"
20  PRINT "MINNESOTA"
30  PRINT "56258"
40  END
```

Exercise 1 - 2

---

1. Write a program to print your full name on a single line. Enter your program to the microcomputer and run it.

2. Write a program to print your first, middle, and last names on separate lines. Enter the program to the microcomputer and run it.

3. Find the errors in this program:

```
100
10+2  PRINT "MY NAME IS"
200   PRINT "JOHNNY"
300   END
400   PRINT "WHAT'S YOURS ?"
```

has to be last one

need input statement

4. Which of the following are legal line numbers?

100	20A	15.5	-32	01000
63000	0	PRINT	A100	+235

5. Describe what the PRINT and END statements do.

NEW
10 PRINT "LEONE"
RUN

5 HOME
10 PRINT"

1 - 3 Printing Results of Calculations

BASIC OPERATORS

In BASIC arithmetic operators instruct the computer to perform a calculation.

BASIC Symbol	Arithmetic Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Raise a number to a power

PRINT AND CALCULATIONS

The PRINT statement can be used for a direct calculation without a line number or as part of a program. Direct calculation:

general form: PRINT any calculation not enclosed in quotes

example: PRINT 82 * 4

purpose: The PRINT statement without a line number causes numeric expressions to be evaluated immediately.

sample: PRINT 4 * 5 + 2

solution: 22

PRINT used in a program:

general form: LN PRINT any calculation not enclosed in quotes

example: 6Ø PRINT 25 + 13 - 21

purpose: The PRINT statement causes numeric expressions, not enclosed in quotes, to be evaluated. The result is printed on the screen.

sample 1: Write a program to calculate the following expressions:

A. 2 X 2 X 2
B. 5 + 3 + 1
C. 1ØØ - 5Ø - 25

solution:

```
1Ø  PRINT 2 * 2 * 2
2Ø  PRINT 5 + 3 + 1
3Ø  PRINT 1ØØ - 5Ø - 25
4Ø  END
```

sample 2: Write a program to calculate the following expressions:

A. 3 raised to the second power
B. 5 raised to the fourth power

solution:

```
1Ø  PRINT 3 ^ 2
2Ø  PRINT 5 ^ 4
3Ø  END
```

RUN - does not have a line Number

Exercise 1 - 3

1. Write BASIC expressions for each of the following mathematical expressions:

   A. 23 X 32 10 PRINT 23 * 32

   B. 2419
      3347
      1989
    + 4434
      20 PRINT 2419+3347+1989+4434

   C. 2 + 4 (3 ÷ 2) 30 PRINT 2+4*(3/2)

   D. $5\ (7)^2 - 3^3$ 40 PRINT 5*7^2-3^3

   E. Ø.3275
      X .521
      50 PRINT .3275 * .521

2. Change the following BASIC expressions to mathematical expressions:

   A. 3 ^ 4 + 6 3^4+6
   B. 5 * (2 + 8) $5\ (2+8)$
   C. 8 * 3 ^ 2 $8(3^2)$

3. Use the direct calculation process on the system to solve the following:

   A. 3 * 4
   B. 6 * -4 + 3 6*(-4)+3
   C. 4 / 2 + 3
   D. 5 ^ 2 + 6

4. Write a BASIC program that prints the messages ADDITION =+, SUBTRACTION =-, MULTIPLICATION =*, DIVISION =/, on four different lines.
use quotes

5. Write a BASIC program to print the answers to:

   A. 7.27 10 PRINT
      X 2.8

   B. 8.71
      X Ø.23

   C. 3.75 * 25.81
   D. 25.81 X 3.75
   E. 3.21 X 2.Ø152 RUN

6. Write a BASIC program to print the message BASIC IS EASY five times.

```
10 PRINT " BASIC IS EASY "
20 PRINT " BASIC IS EASY "
30 PRINT " BASIC IS EASY "
40 PRINT " BASIC IS EASY "
50 PRINT " BASIC IS EASY "
60 END
RUN
```

1 - 4 Order of Operations

Expressions are evaluated in an exact order by the computer known as the Order of Operations. The rules below state the order in which the computer performs these operations. All expressions are evaluated from left to right. If two or more equal value operations appear next to each other, they are performed in their written order from left to right.

	OPERATOR	EXPLANATION
1.	( )	Inside of parenthesis are evaluated from inner set to outer set.
2.	^	Exponentiation - numbers are raised to powers.
3.	* or /	Multiplication or division are evaluated as they occur from left to right.
4.	+ or -	Addition or subtraction are evaluated as they occur from left to right.

Operations must never be left in doubt. An operator always separates numbers, variables, and parenthesis in expressions.

example: Consider the expression: 25 * 4 - 8 / 2 ^ 3

The computer scans the expression from left to right looking first for parenthesis. It doesn't find any. Next it scans for the exponent symbol ^, and calculates 2 ^ 3 = (8). It scans next from left to right looking for multiplication and division. It first calculates 25 * 4 = (100) and next calculates 8 / 8 = (1). Finally, it checks for addition and subtraction. The result 1 is subtracted from the result 100, giving a value of 99 to the expression.

Exercise 1 - 4

1. Write the answer to the following BASIC expressions:

   A. 50 / 5 ^ 2 = 2
   B. 4 + 2 * 6 49 = 16
   C. (5 - 3 2) * 7 ^ 2 = 147
   D. 14 / 2 - 3 * 2 = 1
      7 6

2. Write a BASIC program to print the answers to:

   A. $3(6 + 3) \times 4$ 10 PRINT 3(6+3) * 4
   B. $4^2 \times 6 - 2$ 20 PRINT 4 ^ 2 * 6 - 2

1 - 5 Assigning Values to Variables

NAMING VARIABLES

Variables name a storage location in the microcomputer's memory. Before a number can be scored in the computer, it must be assigned a name. A Numeric Variable in BASIC is named by:

1. A single letter of the alphabet or,
2. A single letter followed immediately by an alphanumeric character. An alphanumeric character is any letter A through Z, or any digit zero through nine. Examples:

   A C9 D2 Z BØ CC FG R2

   A numeric variable name may be up to 238 characters long, but APPLESOFT BASIC uses only the first two characters to distinguish one name from another. Check the APPLESOFT BASIC PROGRAMMING REFERENCE MANUAL for reserved words that cannot be used as variable names or part of variable names.

THE ASSIGNMENT OPERATOR

The = symbol is the assignment operator when used in a LET statement. It causes the number or value of the expression to its right to be assigned to the variable on its left.

example: 1Ø LET A = 1ØØ

THE ASSIGNMENT STATEMENT

general form: LN LET v = e

where: LN is a legal line number

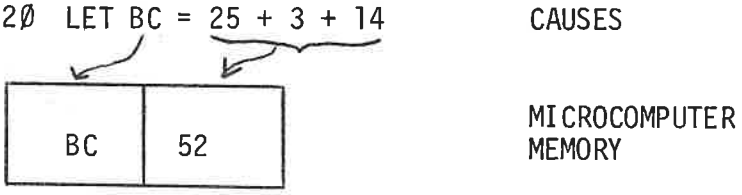
v is a legal numeric variable

e is any mathematical expression

example:

```
1Ø  LET A = 1Ø
2Ø  LET B = 25 + 13 * 14
```

purpose: The LET statement assigns a value to a variable for later reference in the program. When a LET statement is executed, the computer locates an empty storage cell, gives it the name of the variable, evaluates the expression, and places the resulting value in that storage location. That value can be referenced later in the program by the variable name.



PRINTING THE VALUE OF A VARIABLE

general form: LN PRINT v OR LN PRINT e

example: 9Ø PRINT K OR 95 PRINT B4 + 1Ø

purpose: The PRINT statement in this form prints the value of a variable. Variables may also appear in expressions in PRINT statements. In the second example, B4 + 1Ø is first evaluated; then the result is printed. Spaces in an expression like B4 + 1Ø are disregarded by the microcomputer.

sample 1: Write a BASIC program that assigns .5 to the numeric variable BK and assigns 6.5 to the numerical variable D2. Assign the product of the two to the numeric variable C3. Print out the answer.

solution 1:

```
1Ø  LET BK = 5
2Ø  LET D2 = 6.5
3Ø  LET C3 = BK * D2
4Ø  PRINT C3
5Ø  END
```

Exercise 1 - 5

---

1. Which of the following are legal variable names in BASIC?

   A. CN
   B. 5A
   C. L
   D. W9
   E. H2
   F. 5V
   G. 22
   H. AA

2. Which of the following are legal line numbers?

   A. Ø
   B. 125
   C. -25
   D. 64ØØØ
   E. +4

3. What are the BASIC symbols used for the following arithmetic operations?

   A. Multiplication
   B. Division
   C. Raising a number to a power

4. Write BASIC expressions for each of the following mathematical expressions:

   A. 2 + 4 (3 / 2)
   B. 9 - 3
   C. 9 / 5 (1ØØ) + 32

5. What symbol is used to assign numbers to variables?

6. Write statements which would assign the following numbers to variables of your choice:

   A. 325
   B. -75
   C. 12.3456
   D. 2^3

7. Write BASIC statements to calculate the following:

   A. 1 / 3 + 1 / 4 + 1 / 5 + 1 / 6
   B. 1 - 1 / 2 + 1 / 3 - 1 / 4 + 1 / 5
   C. 2 ^ Ø + 2 ^ 1 + 2 ^ 2 + 2 ^ 3 + 2 ^ 4
   D. 5 / 9 * (-4Ø - 32)

8. Write a complete BASIC program to:

   A. Assign the number 212 to the variable F
   B. Calculate the expression 5 / 9 * (F - 32) and assign the result to the variable C
   C. Terminate the program

9. Enter the program in Exercise 8 and have the microcomputer run it. What happens? What should be added to the program?
   NOTHING

10. Write a complete BASIC program to:

   A. Assign the length 24 to the variable L
   B. Assign the width 16 to the variable W
   C. Assign the height 8 to the variable H
   D. Calculate the value of the expression L * W * H and assign the result to the variable V
   E. Print the result V and terminate the program

NEW

```
10  F = 212
30  C = 5/9*(F-32)
40  PRINT C
50  END
60  RUN
```

C = 100

use 10 INPUT F

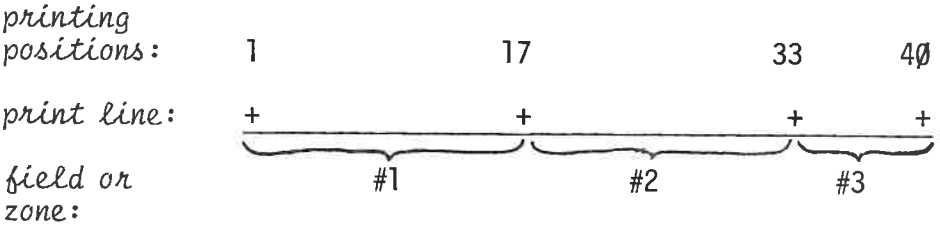
TO PUT ON DISK
SAVE SPACE PROGRAM NAME

Read Chapter 2
P. 29

1 - 6 Printing More than One Item on a Line

THE PRINT FIELD

Two or more items may be printed on one line by separating the items with a comma. The comma causes the print line, consisting of 40 characters, to be divided into three fields or print zones.



The first two zones contain 16 available printing positions. The third zone contains 7 available printing positions. Additional print zones will be printed on the next lines.

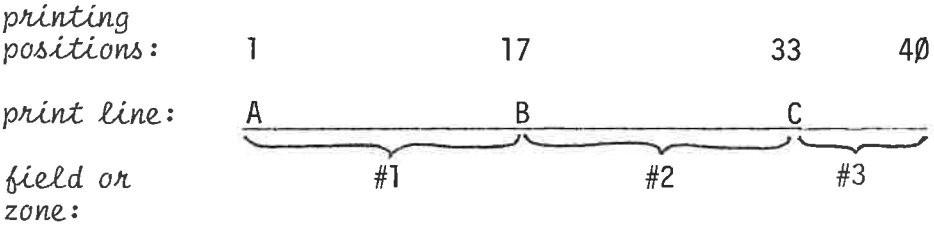
Commas serve a function similar to the TAB key on a typewriter. Whenever a comma is encountered outside of quote marks in a PRINT statement, the print carriage moves to the beginning of the next available print zone.

PRINTING MORE THAN ONE MESSAGE

example 1:

```
20 PRINT "A", "B",  "C"
```

result:



Notice that each message begins in a separate zone. The comma causes the print carriage to move to the beginning of the next available print zone.

example: 3Ø PRINT "A", , "B"

result:

```
printing
positions:    1               17               33      4Ø

print line:   A                                B
              ________________________________________
field or             #1               #2          #3
zone:
```

This time, two commas occurred between each message. Two commas in a row cause a print field to be skipped entirely.

example 3: 4Ø PRINT "ABCDEFGØ123456789", "B"

result:

```
printing
positions:    1               17               33      4Ø

print line:   ABCDEFGØ123456789
              ________________________________________
field or             #1               #2          #3
zone:
```

In this example, a message filled one print zone and carried into the next. The comma directs the computer to print the following message at the beginning of the next available print zone.

<u>MIXING ITEMS</u>

Messages, variables, and computations may be freely mixed in any order in a PRINT statement.

example 4: 6Ø PRINT "25 + 25 = ", 25 + 25

result: 25 + 25 = 5Ø

example 5: 6Ø PRINT "C = ", 5 / 9 * (5Ø - 32)

result: C = 1Ø

These examples illustrate how to combine a message and a computation.

PACKING

The semi-colon (;) may also be used to separate items in a PRINT statement. Encountering a semi-colon outside of quote marks, the computer causes the print carriage to remain in its present position and begins printing the next item at that location. Use of the semi-colon is referred to as packing information.

example 6: 1Ø PRINT "A= ";25

result:

printing positions: 1 2 3 4 5 6 7 8 9 . . . . . .

print line: A = 2 5

Notice that a space had to be typed after the equal sign for a space to appear on the PRINT Line. Care must be taken using the semi-colon to include needed spaces in the messages.

example 7: 2Ø PRINT "PROBLEM 5", "THE ANSWER IS "; 29

result:

printing positions: 1 17

print line: PROBLEM 5 THE ANSWER IS 29

field or zone: #1 #2

Commas and semi-colons may be freely mixed in a PRINT statement. A PRINT statement ending with a semi-colon as the last character, causes the print carriage to freeze when done printing. The first item in the next PRINT statement begins in that position. This allows more than one PRINT statement to print on a single line.

example 8:

```
10  PRINT "A=";
20  PRINT 10
```

result:

printing positions: 1 2 3 4 5 6 7 8 9 . . . . . .

print line: A = 1 0

Exercise 1 - 6

1. The comma causes the print line to be divided into _____ zones.

2. Give an example of a PRINT statement using commas.

3. Give an example of a PRINT statement using semi-colons.

4. Using one PRINT statement, write a program to print your:

   A. First name in print zone 1
   B. Middle name in print zone 2
   C. Last name in print zone 3

5. Using one PRINT statement, write a program to print your:

   A. First name in print zone 1
   B. Last name in print zone 3

6. Write a program to:

   A. Assign the value 10 to F
   B. Using the formula C = 5 / 9 * (F - 32) calculate a value for C
   C. Print the message C = followed by its value

7. Write a program to perform the following temperature conversions and print the results with an appropriate message. Use the formulas:

   F = 9 / 5 * C + 32 and C = 5 / 9 * (F - 32)

   A. Find C when F = 212
   B. Find F when C = 0
   C. Find F when C = -40
   D. Find C when F = -40

8. Write a program to find the perimeter for each of the following rectangles. Use the formula:

   P = 2 X L + 2 X W

   A. L = 9.23, W = 5.96
   B. L = 1095.41, W = 865.31
   C. L = 19.9, W = 16.7

9. Messages can be very useful when printing tables. The message is used as a heading to inform the reader what the columns mean. Write a program to:

   A. Print the headings NUMBER and SQUARE
   B. Print, in the proper columns, the numbers from 1 to 5 and their squares

1 - 7 Entering Information While the Program is Executing

THE INPUT STATEMENT

general form: LN INPUT v

LN INPUT v1, v2, . . .

where: v, v1, v2 are legal numeric variables

purpose: INPUT is a key word instructing the computer to print a question mark and wait for the user to type in a number for each variable in the statement.

The INPUT statement allows numbers to be assigned to variables while the program is being executed. This statement causes a question mark to be printed, the program to halt, and the computer to wait for the user to type in a number for each variable in the statement. The computer then assigns the number(s) typed, to the variable(s), and continues executing the program.

APPLICATIONS FOR THE "INPUT" STATEMENT

The INPUT statement is necessary for programs which ask the user to respond to questions. INPUT is the only statement which allows you to type information at the terminal while a program is executing.

sample 1: Write a program to find the perimeter of two rectangles using INPUT statements.

A. Length = 1Ø, width = 1Ø
B. Length = 15, width = 7

solution 1:

```
1Ø  INPUT L
2Ø  INPUT W
3Ø  PRINT 2 * (L + W)
4Ø  INPUT L
5Ø  INPUT W
6Ø  PRINT 2 * (L + W)
7Ø  END
```

sample run:

```
?  1Ø
?  1Ø
   4Ø
?  15
?   7
   44
```

Notice the question marks in the sample run. Each time the INPUT statement is executed, a question mark is typed by the terminal. The computer waits for a number to be entered. The variables L and W can both appear in the same INPUT statement saving programming time and typing effort as in the following solution.

solution 2:

```
10  INPUT L,W
20  PRINT 2 * (L + W)
30  INPUT L,W
40  PRINT 2 * (L + W)
50  END
```

sample run:

```
?  10,10
   40
?  15, 7
   44
```

Notice in this example two variables were used in the INPUT statements. INPUT still caused a single question mark to appear. Fewer lines were needed in both writing the program and in the printing on the screen.

DOCUMENTING INPUT STATEMENTS

Suppose you were told to execute a program on the computer. After typing RUN a question mark appeared. What do you type in? Unless someone gave you instructions, or unless you wrote the program, it would be difficult to continue. Programs using INPUT statements should contain explanatory messages. They should be printed before the INPUT statement is executed. These messages serve a very important purpose. They tell the person using the program what and how to respond to the question mark.

sample: Write instruction messages for the perimeter program

solution 1:

```
 5  PRINT "ENTER A LENGTH AND WIDTH
10  INPUT L,W
```

```
20  PRINT 2 * (L + W)
25  PRINT "ENTER A LENGTH AND WIDTH"
30  INPUT L,W
40  PRINT 2 * (L + W)
50  END
```

sample run:

```
    ENTER A LENGTH AND WIDTH
?   10,10
    40
    ENTER A LENGTH AND WIDTH
?   15,7
    44
```

Notice how the message makes it clear what the user is to type. Let's examine messages once more. Messages make the printing understandable and easy to interpret. However, it still may not be clear to the user what the number 40 means in the above example. In addition, single space printing may be difficult to read.

solution 2:

```
 1  PRINT "THIS PROGRAM CALCULATES PERIMETERS FOR RECTANGLES."
 5  PRINT "ENTER A LENGTH AND WIDTH"
10  INPUT L,W
15  PRINT
20  PRINT "THE PERIMETER IS "; 2 * (L + W)
25  PRINT
30  PRINT "ENTER A LENGTH AND WIDTH"
31  INPUT L,W
35  PRINT
40  PRINT "THE PERIMETER IS "; 2 * (L + W)
50  END
```

sample run:

```
    THIS PROGRAM CALCULATES PERIMETERS FOR RECTANGLES
    ENTER A LENGTH AND WIDTH
?   10,10
    THE PERIMETER IS 40
    ENTER A LENGTH AND WIDTH
?   15,7
    THE PERIMETER IS 44
```

The use of the PRINT statement as in lines 15, 25, and 35 cause a blank message to be printed. This is the same as skipping a line or double spacing. It serves the purpose of making the printout easier to read.

The first line in the sample run, THIS PROGRAM CALCULATES PERIMETER OF RECTANGLES makes it clear to the person using the program what its purpose is. The message, THE PERIMETER IS, clarifies what the numbers being printed by the computer mean. Using messages in this manner is called documenting a program.

Notice that the question mark is typed on the next line. The question mark will be printed on the same line if a semi-colon is typed at the end of the line prior to the INPUT line.

solution 3: Change lines 5 and 3Ø to read:

```
 5  PRINT "ENTER A LENGTH AND WIDTH";
3Ø
```

INCORRECT RESPONSE TO INPUT

If the information typed does not match the INPUT statement requirements, the computer will respond to the users entry in one of the following ways:

A. ?? -- Not enough data was entered. Enter the necessary data after the double question marks.

B. REENTER -- Incorrect data was entered. Reenter the data.

C. EXTRA IGNORED -- Too much data was entered and the extra was ignored.

SPECIAL FORM OF INPUT

When using several lines of instructions for a program it becomes necessary to hold part of the instructions on the screen until the person executing the program is ready for more instructions.

THE GET STATEMENT

general form: LN GET A$

where: A$ is the special variable used with the GET statement

purpose: The GET statement fetches a single character from the keyboard without displaying it on the screen and without requiring that the RETURN key is pressed.

APPLICATION FOR THE "GET" STATEMENT

The GET statement is necessary in programs where the instructions of the program will fill more than one screen of information. A screen is twenty-four lines of information.

sample:

```
10  PRINT "THIS PROGRAM WILL DRILL THE STUDENTS"
20  PRINT
30  PRINT "ON THE 50 STATES IN THE UNITED STATES."
40  PRINT
50  PRINT
60  PRINT
70  PRINT "PRESS SPACE BAR TO CONTINUE"
80  GET A$
```

Exercise 1 - 7

1. What statement causes a question mark to be printed?

2. What statement causes the terminal to skip one line?

3. Which of the following are correct BASIC statements?

```
1Ø  PRINT "THE PERIMETER = ", INPUT
2Ø  PRINT
3Ø  INPUT A:B:C
4Ø  PRINT "ENTER THE NUMBER OF INCHES IN A FOOT"
5Ø  INPUT I
```

4. Describe what happens when the computer executes this statement:

```
1ØØ  PRINT A
```

5. When is it necessary to use an INPUT statement?

6. Write INPUT statements to do the following:

   A. Ask for a number to be assigned to A
   B. Ask the user to type two numbers. The first will be assigned to L and the second assigned to W
   C. Ask for four numbers to be assigned to C1, C2, C3, and C4

7. Write a PRINT statement giving instructions for entering the length, width, and height of a room. Use a semi-colon at the end of the statement so the question mark is printed on the same line. Next, write a statement which allows the length 12, the width 9, and the height 8, to be entered while the program is executing. Finally, write a PRINT statement which calculates the volume of the room. Use the formula:

   V = L X W X H

8. When should GET A$ be used in a program?

9. Use an INPUT statement to write a program that requests a person's age. Include a PRINT statement that includes YOUR AGE IS followed by the variable.

1Ø. Suppose you have been told you will have to convert inches to centimeters several times over the next two weeks, and you decide to write a program to do the conversion for you. Each time you run the program it will complete one conversion. To write the program use the formula:

   C = 2.54 X I

Review - Chapter I

Now that you have completed Chapter I, you should be able to write simple BASIC programs, enter them into the microcomputer and run them. Let's review the important topics covered in this chapter before proceeding to the chapter quiz.

STRUCTURE OF A BASIC STATEMENT

LN KEYWORD PARAMETERS

LN (line numbers) -- Every BASIC statement must have a line number from 1 and 63999. Line numbers determine the order statements are executed in. Numbering statements in increments of 1Ø or 2Ø gives you room to insert additional statements and make corrections.

KEYWORD -- The KEYWORDS in a statement tell the computer what action is to be performed. The KEYWORD follows the line number.

PARAMETER -- The PARAMETER portion of a statement contains the data for the computer to act on or to perform a calculation on.

BASIC STATEMENTS INTRODUCED

PRINT -- The PRINT statement is the most important of the BASIC statements. It allows the programmer to:

A. Print messages
B. Print results to computations
C. Print values assigned to variables
D. Print combinations of A, B, and C
E. Automatically positions to columns called zones, when using the comma as a separator
F. Skips a line (or double spaces) when the parameter portion of the statement is omitted

END -- The END statement instructs the computer that this is the last statement in the program.

LET -- The LET statement assigns numbers to variables and evaluates expressions, assigning the result to a variable. The values are determined at the time the program is written.

INPUT -- The INPUT statement assigns a number typed at the terminal to a variable while the program is executing. INPUT statements usually

require documentation by printing messages before they are executed. GET is a special form of input that fetches a single character from the keyboard without displaying it on the screen and without requiring that the RETURN key be pressed.

BASIC OPERATORS

The symbols used in BASIC to perform computations are:

+	addition
-	subtraction
*	multiplication
/	division
^	raise a number to a power

ORDER OF OPERATIONS

The computer performs computations by these rules:

First -- computations within parenthesis

Second -- numbers are raised to powers

Third -- any multiplication or division is performed in order from left to right

Fourth -- addition or subtraction computations are performed in order from left to right

Review Quiz - Chapter I

1. Which of the following are not legal line numbers?

   A. 6.4
   B. 462
   C. +362
   D. -4(2)
   E. 1,236
   F. -4
   G. 5.Ø12
   H. 1Ø323

2. Which of the following are not legal variable names?

   A. A3
   B. B*
   C. C=
   D. END
   E. 2D
   F. Z
   G. 37
   H. PI
   I. F2
   J. CAT

3. List each of the arithmetic operators used in BASIC and state the operation each performs.

4. List the order of operations performed by the computer.

5. Which operations would be performed first in the following expressions?

   A. A + C * (9 - B)
   B. 5 * 2 ^ 3
   C. A + 2 - C

6. Which of the following do <u>not</u> mean X multiplied by Y in the BASIC language?

   A. X * Y
   B. Y * X
   C. XY
   D. X x Y
   E. X · Y
   F. X (Y)

7. Assuming (A = 2, B = 3, and C = 4), evaluate each of the following BASIC expressions:

   A. A ^ B
   B. A / B * C
   C. A - B / A + C
   D. (A + B) / (B + A)
   E. A * B / (A + C)

8. What is the value of S after the execution of the following BASIC statements?

```
1Ø  LET A = 4
2Ø  LET B = 3
3Ø  LET C = 5
4Ø  LET X = A + B + C
5Ø  LET S = X / 3
```

9. Which of the following statements are false?

   A. Every BASIC program must have an END statement.
   B. Every BASIC statement must have a line number.
   C. Every BASIC program must have one LET statement.
   D. Every BASIC statement must have an equal sign.

10. Write a LET statement for each of the following situations:

    A. Assign the value of 347 to the variable H.
    B. Assign the value of the expression (B + C) to the variable Y.
    C. Multiply A by 5 and assign the result to M.
    D. Assign the value of the expression 9 / 5 * C + 32 to F.

11. Change each of the following LET statements to INPUT statements:

    A. 10 LET B = 0
    B. 20 LET I = 10
    C. 30 LET M = 5

12. State the purpose of an INPUT statement.

13. Write a PRINT statement to accomplish each of the following:

    A. Print the message IT IS A NICE DAY
    B. Print the result of 5 - 4 + 2 * 1
    C. Skip a line
    D. Print the message THE ANSWER TO 9 TO THE SECOND POWER IS followed by the correct answer.

14. What is the purpose of the END statement? What must be true of its line number?

15. What is the purpose of a LET statement?

16. Describe what is meant by the term PRINT FIELD?

17. The comma and the semi-colon have special uses in PRINT statements. Describe what happens when each is used.

18. What is meant by documenting a program?

19. Write a PRINT statement to print the letter A in print position 1 and the letter B in print position 33.

20. Write a program to print your name, address, and zip code as it might appear on a letter envelope.

21. Write a program to print a table of the numbers from 1 to 5, their squares, and their cubes.

22. Write a program to print a table of temperatures in Fahrenheit and their corresponding values in Centigrade. Use the temperatures for Fahrenheit below:

    A. 0
    B. 32
    C. 100
    D. 180
    E. 424

C = 5 / 9 * (F - 32)

USE INPUT + LET STATEMENTS

23. Write a program for calculating the area of a rectangle. The numbers should be entered while the program is being executed. Allow for the calculating of two different areas.

24. A formula to change kilograms to pounds is P = 2.2 X K where P is pounds and K is kilograms. Write a program to allow another student to type in any value in kilograms and to get back the correct number of pounds.

Chapter II

LOOPING

2 - 1 Replacement

The statement "1ØØ LET A = 1Ø" may be called a replacement statement. When executed by the computer, the statement instructs the computer to:

1. Calculate a value for the expression to the right of the assignment operator.
2. Locate the storage cell named or addressed by the variable to the left of the assignment operator and erase the value stored there.
3. Write the value of the expression in the storage cell named by the variable to the left of the assignment operator.

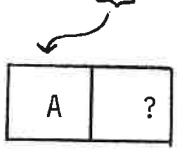
The statement is called a REPLACEMENT statement because the old value in the named storage location is replaced by the new value. Thus:

1ØØ LET A = 1Ø — instructs the computer to evaluate the expression to determine what number is to be written

and

1ØØ LET A = 1Ø — instructs the computer to write this value into the named storage cell

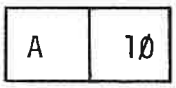
1ØØ LET A = 1Ø — storage cell is identified



MICROCOMPUTER MEMORY

location value

```
100  LET A = 10
```



location value

a number is written into that cell replacing any previous value

Now, examine a second example:

```
190  LET A = 1
200  LET B = A + 1
```

```
190  LET A =  1
```

instructs the computer to determine what number is to be written

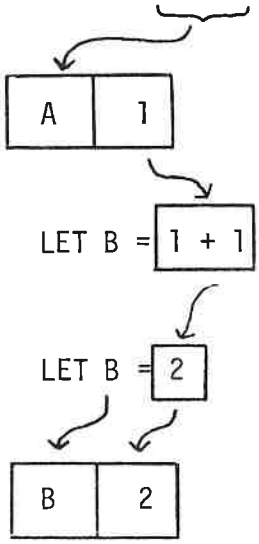
```
190  LET A    = 1
```

causes the storage cell named on the left of the assignment operator to be located, and the value of the expression to be written into it

In the second statement a variable appears in the expression.

```
200  LET B = A + 1
```

CAUSES:



a storage location is identified

the value stored there is substituted for the variable in the expression

a value is calculated

the result replaces the previous value in the storage location

Note: Only the storage location named to the left of the assignment operator is changed. The value stored in A remains the same.

BEFORE EXECUTION OF LINE 2ØØ

A	1
B	?

AFTER EXECUTION OF LINE 2ØØ

A	1
B	2

COUNTING

We can instruct the computer to count using replacement.

example:

```
1Ø  LET A = Ø
2Ø  LET A = A + 1
3Ø  LET A = A + 1
```

causes:

```
1Ø  LET A = Ø      —— causes →
2Ø  LET A = A + 1  —— causes →
3Ø  LET A = A + 1  —— causes →
```

A	Ø

A	1

A	2

COMPUTER MEMORY

Counting is an important programming tool which will be used in problems throughout this text.

sample 1: Write a program to print the counting numbers from one to five.

solution:

```
1Ø  LET C = 1
2Ø  PRINT C
3Ø  LET C = C + 1
4Ø  PRINT C
5Ø  LET C = C + 1
6Ø  PRINT C
7Ø  LET C = C + 1
8Ø  PRINT C
9Ø  LET C = C + 1
1ØØ  PRINT C
11Ø  END
```

Notice in this program statements 3Ø, 5Ø, 7Ø, and 9Ø are the same. Each

of these statements cause 1 to be added to storage location C. Also note that statements 2Ø, 4Ø, 6Ø, 8Ø, and 1ØØ are the same.

sample 2: Write a program that will count by fives to fifteen.

solution:

```
1Ø  LET N = 1
2Ø  LET M = 5
3Ø  LET T = N * M
4Ø  PRINT T
5Ø  LET N = N + 1
6Ø  LET T = N * M
7Ø  PRINT T
8Ø  LET N = N + 1
9Ø  LET T = N * M
1ØØ  PRINT T
11Ø  END
```

This program prints multiples of five. The program could easily be modified to obtain multiples of any number. Which statement would be changed to obtain multiples of seven?

2Ø LET M = 7

Exercise 2 - 1

1. Describe why a LET statement is also called a REPLACEMENT statement.
the old value in a named storage location is replaced by a new value
2. The variable G in the statement: 15 LET G = 15 * 3

   A. Determines an address in the microcomputer's memory
   B. Will have the old value replaced by 45
   C. May be used later in the program to represent the number 45
   D. All of the above

3. Write a BASIC program to print the multiples of three between 15 and 27.

4. Write a BASIC program that counts backwards by fours from 67 to 35. Print each number as it counts.

5. Write a program that will count by multiples of any number. Use the INPUT statement to input the starting number. Print out the first three multiples.

TABLES CAN BE CREATED EASILY BY USING THE COUNTING TECHNIQUE

6. Write a program to print a table of numbers from 1 to 1Ø5 and their square roots. (A square root is calculated by raising a number to the .5 power.)

7. Prepare a program that will convert inches to centimeters. (Hint: 1 inch = 2.54 centimeters.) Have the program print columns with headings and show the comparison of 7, 14, 21, 28, and 35 inches to centimeters.

2 - 2 The Uncontrolled Loop

In section 2-1 the problem of counting by fives required three statements to be repeated. Suppose you were asked to write a program to count from 1 to 1ØØ. Your program might start like this:

example:

```
1Ø  LET C = 1
2Ø  PRINT C
3Ø  LET C = C + 1
4Ø  PRINT C
5Ø  LET C = C + 1
```

These statements demonstrate the tedious task of writing such a program. The pair of statements LET C = C + 1 and PRINT C would have to be repeated for each number printed. The GO TO statement allows other statements to be repeated over and over without writing them each time.

THE GO TO STATEMENT

general form: LN GO TO m

where: m is the line number of a statement in your program

example:

```
4Ø  GO TO 2Ø
```

purpose: The GO TO statement causes the normal execution of the program to be broken and program control to be transferred to the statement whose line number is given.

sample 1: The task of writing a program to count from 1 to 1ØØ can be greatly shortened.

solution:

```
1Ø  LET C = 1
2Ø  PRINT C
3Ø  LET C = C + 1
4Ø  GO TO 2Ø
5Ø  END
```

Repeating the execution of statements 2Ø, 3Ø, and 4Ø in this example is

called LOOPING. The statements which are repeated form a LOOP.

If you enter this program and run it on the microcomputer, you will notice it starts printing at number 1 and continues on until you stop it. The computer is never allowed to reach the END statement in the program. Since the END statement is never executed, the program is said to contain an INFINITE LOOP.

The only way to stop this program while it is executing is to hold down the control (CTRL) key and type a C. This terminates programs which are printing information.

Note: LOOPS are statements in a program that are repeated over and over. INFINITE LOOPS are caused by repeating statements in such a way that the END statement is never executed.

sample 2: Here is a second example of a program with an infinite loop.

solution:

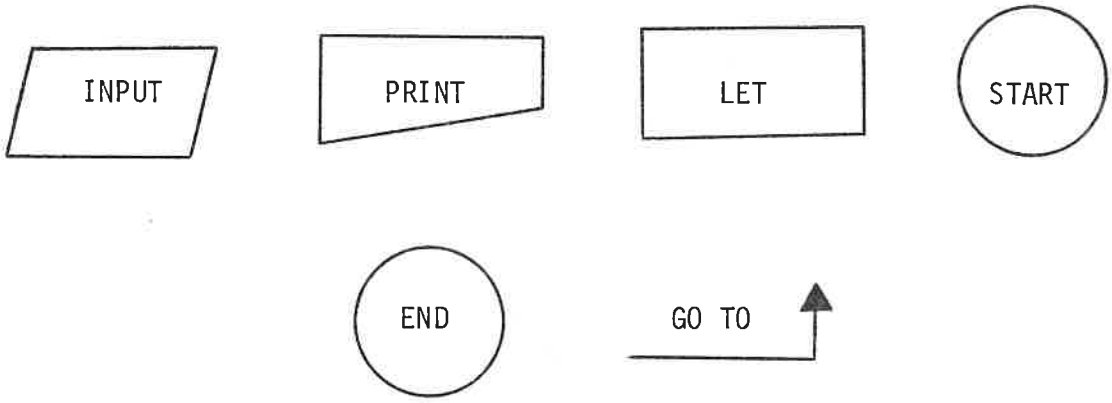
```
10  LET A = 1
20  LET B = 2
30  PRINT "WHAT IS THE SUM ";A;"+";B
40  INPUT C
50  LET A = A + 1
60  LET B = B + 2
70  GO TO 30
80  END
```

The statements 30, 40, 50, 60, and 70 are being repeated over and over. The END statement is never reached. This time, however, the INPUT statement at line 40 causes a break in the printing and waits for the user to enter a number. To stop a program at this point while pressing the CTRL key, type the letter C, release the CTRL key and C key and press the return key. This directs the computer to terminate the program.

NOTE: To stop or terminate programs containing infinite loops, a control C must be entered. Hold the CTRL key down and type a C, then press the RETURN key.

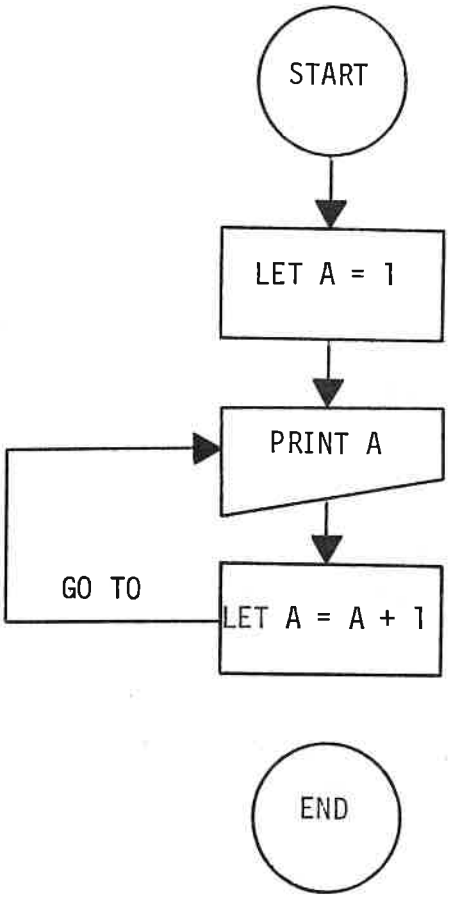
DETECTING LOOPS

Flowcharting is a useful tool for detecting loops. It is a method of showing the order statements are executed. Special symbols and arrows are used to draw a map of your program's execution.



A flowchart is read by beginning at the START symbol and following the arrows to the END symbol.

Here is a flowchart for printing the counting numbers from 1 to . . . forever.



Infinite loops are easily detected by examining flowcharts for programs. In the example above, the END statement does not have an arrow leading to it. Therefore, it cannot be executed.

Exercise 2 - 2

1. Describe what is meant by:

   A. A loop
   B. An infinite loop

2. How do you stop a program with an infinite loop which is:

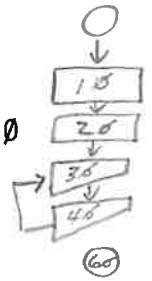
   A. Executing a PRINT statement
   B. Executing an INPUT statement

3. What symbols are used in a flowchart for:

   A. A PRINT statement
   B. The beginning of the flowchart
   C. A GO TO statement
   D. An INPUT statement

4. Draw a flowchart for:

```
1Ø  LET CAT = 1Ø
2Ø  LET DOG = CAT * 1Ø
3Ø  PRINT CAT
4Ø  PRINT DOG
5Ø  GO TO 3Ø
6Ø  END
```



5. Draw a flowchart for:

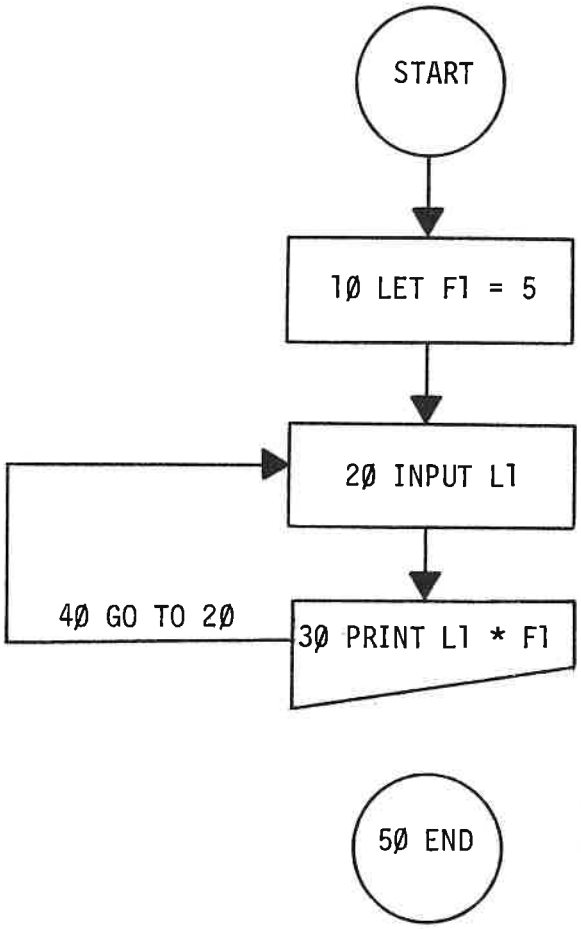
```
1Ø  LET C = 1
2Ø  PRINT "I AM A PROGRAMMER"
3Ø  LET C = C + 1
4Ø  PRINT C
5Ø  END
```

6. Does the program in problem 4 or 5 have a loop? An infinite loop? If so, what statement(s) are being repeated? How would you stop the program?

7. Draw a flowchart to:

   A. Start at 1ØØ and count backwards by 1Ø's.
   B. Do you have an infinite loop?

8. Write a program for this flowchart:



9. Draw a flowchart for the following program:

```
1Ø  PRINT "ENTER NUMBER OF MILES DRIVEN"
2Ø  INPUT M
3Ø  PRINT "ENTER THE NUMBER OF GALLONS OF GAS USED";
4Ø  INPUT G
5Ø  LET N = M / G
6Ø  PRINT "YOU OBTAINED ";N;" MILES PER GALLON."
7Ø  GO TO 1Ø
8Ø  END
```

1Ø. Study the flowchart you have drawn for problem 9:

A. Will the END statement be executed?
B. Which statements are repeated?
C. How would you stop this program?
D. Enter and run the program in problem 9.

1) The first time through the program enter numbers of your choice.
2) The second time through the question, enter a Ø when asked for the number of gallons of gas. What happened? Think what you could do to prevent this from happening.

2 - 3 Controlling Loops

Loops can be controlled in a program by an IF-THEN statement. This statement contains a relational expression which asks the question -- Is this condition TRUE or FALSE? What this means is that the computer can make decisions which have a YES or NO solution.

Relational Operators

A relational operator describes the decision to the computer.

Relational Operator	Decision
=	is equal to
<	is less than
>	is greater than
= <	is equal to or less than
= >	is equal to or greater than
< >	is either less than or is greater than

Relational Expressions

The relational expression states the decision to be made. It consists of two numeric expressions separated by a relational operator. The computer evaluates each expression and then compares the value to the left of the operator with the value to its right. A value of TRUE or FALSE is then assigned to the entire expression.

examples:

```
A = B
9 / 5 + 4 > 5 / 9 * 2
A + 5 < B - 3
```

THE DECISION STATEMENT

Programs can avoid infinite loops by using IF-THEN statements as a test when the program should exit a loop.

general form: LN IF (relational expression) THEN m

where: m is the line number of a statement in your program

example:
```
10 LET A = 35 THEN 20
11 IF B =>C + 3 THEN 100
12 IF 5 =<A * 2 THEN 23
```

purpose: The IF-THEN statement is used to make simple decisions which have YES or NO answers. When executed, the computer evaluates the relational expression for a TRUE or FALSE condition.

If a value of TRUE is assigned to the expression the THEN m portion of the statement is executed. This causes program control to be transferred to the statement whose line number is m.

If a value of FALSE is assigned to the expression, the THEN portion of the statement is ignored and the next statement following the IF-THEN is executed. By using the IF-THEN statement with a counter, a loop can be controlled or executed the desired number of times.

sample: Write a program to print from 1 to 100.

solution:
```
10 LET C = 1
20 PRINT C
30 LET C = C + 1
40 IF C <= 100 THEN 20
50 END
```

Statement 40 controls how many times the loop is executed (statements 20, 30, and 40). As long as the value of C is less than or equal to 100, the loop will be repeated. Once the number 100 has been printed, the relational expression "C <= 100" becomes false and the END statement is executed.

SECOND FORM OF THE IF-THEN STATEMENT

general form: LN IF (expression) THEN (instruction)

where: Instruction is any legal BASIC statement

purpose: This type of IF-THEN can eliminate many statements. This will create a shorter program that is easier to follow. The statement is evaluated the same as the other IF-THEN.

sample 1:

```
10  LET A = 3
20  LET B = 4
30  IF A = B THEN 50
40  LET C = A + B
45  GO TO 60
50  PRINT "SAME"
60  END
```

sample 2:

```
10  LET A = 3
20  LET B = 4
30  IF A = B THEN PRINT "SAME"
40  LET C = A + B
50  END
```

You will notice that the second example will have the same result as the first but is somewhat shorter. If A = B, the word SAME will be printed directly in the second example. In the first example, the program branches to line 50 and then prints the message.

UNCONTROLLED LOOPS REVIEWED

An uncontrolled loop is a loop which is never allowed to terminate. It may also be called an "infinite loop."

loop:

```
10  PRINT "HELLO"
20  LET B = B + 1
30  GO TO 10
40  END
```

In this example, there is no control over the number of times the loop is repeated. The user will have to take some special action to terminate the program.

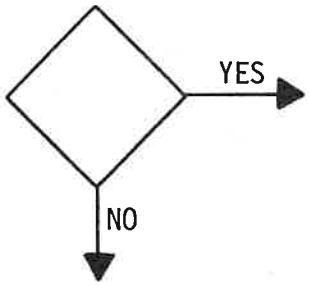
Uncontrolled loops are caused either by using a GO TO statement to repeat the loop or by accidentally using an improper IF-THEN statement.

```
1Ø  PRINT "POOR IF-THEN"
2Ø  LET B = B + 1
3Ø  IF B = > 1 THEN 1Ø
4Ø  END
```

Note, in statement 3Ø, the value of B will always be equal to or greater than 1. Thus, the loop will never terminate.

UNDERSTANDING IF-THEN STATEMENTS

Flowcharts make IF-THEN statements easy to understand. The symbol for an IF-THEN statement is:

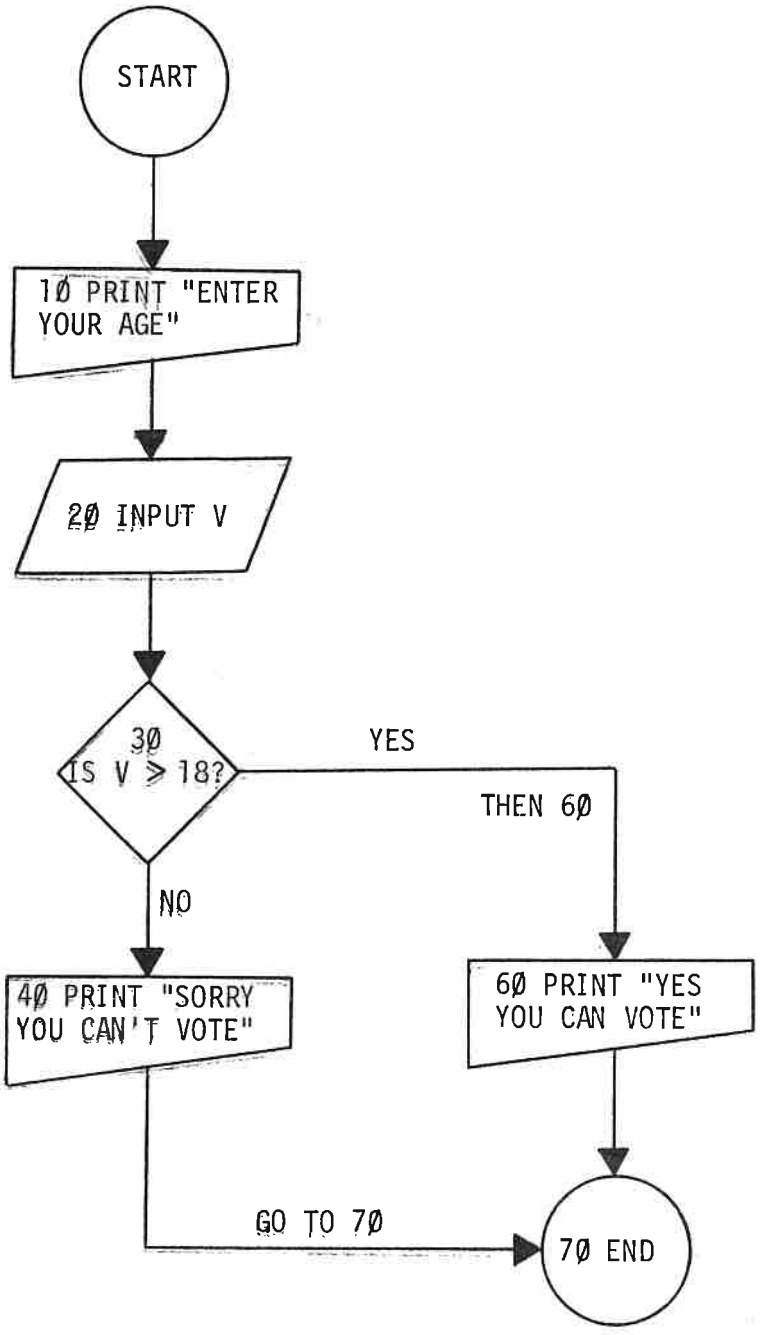


The question being asked is placed inside the symbol. A YES arrow leads from the symbol to the box containing the statement to be executed when the question is answered YES. This corresponds with the THEN m portion of the IF-THEN statement.

The NO arrow leads to the symbol for the statement which is to be executed when the answer to the question is NO. This corresponds to a FALSE IF-THEN condition.

Examine the following program and flowchart. Here is a simple program to determine if a person can vote on Election Day.

```
1Ø  PRINT "ENTER YOUR AGE"
2Ø  INPUT V
3Ø  IF V > 18 THEN 6Ø
4Ø  PRINT "SORRY, YOU CANNOT VOTE"
5Ø  GO TO 7Ø
6Ø  PRINT "YES, YOU CAN VOTE"
7Ø  END
```



TRACING

The tracing of a program means going through the program by evaluating the variables and finding what will be printed at the monitor. The flowchart can be used to trace a program. You will need paper and pencil when tracing and reading a flowchart. Printing at the terminal screen should be printed in a column on the right side of the paper. The values assigned to variables should be listed in columns on the left side of the paper. As a new value is assigned to a variable, the new value should be written down and the old value crossed out.

sample: Write a program to count from 1 to 1ØØØ printing as it counts.

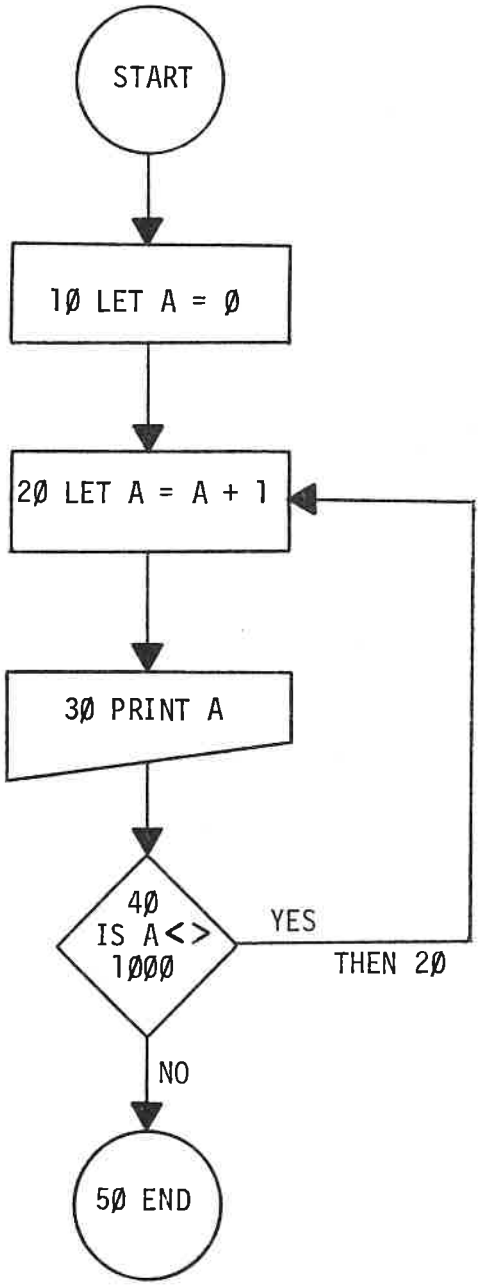
```
1Ø  LET A = Ø
2Ø  LET A = A + 1
3Ø  PRINT A
4Ø  IF A <> 1ØØØ THEN 2Ø
5Ø  END
```

trace variable:

```
 A
 Ø
 1
 2
 3
 4
 .
 .
 .
 .
1ØØØ
```

screen printout:

```
 1
 2
 3
 4
 .
 .
 .
 .
1ØØØ
```



This flowchart has a loop. Lines 2Ø, 3Ø, and 4Ø will be repeated until the value of A is equal to 1ØØØ. The IF-THEN statement controls the loop through a counting variable A. Care must be taken to test for the proper value of the variable A.

Exercise 2 - 3

1. Match the relational operators in the left column with their meanings in the right column:

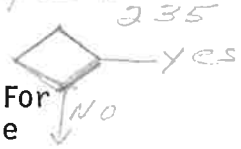
Relational Operator	Decision
=	is less than
<	is equal to or less than
>	is not equal to
= <	is equal to
= >	is equal to or greater than
< >	is greater than

2. Write the question being asked by each of these IF-THEN statements:

A. 4Ø IF A < 5 THEN 4Ø If A is less than 5 then Do to 40
B. 4Ø IF C + D = 7 THEN 1ØØ If C plus D equals 7 then Do to 100
C. 4Ø IF A = B THEN 2Ø If A equals B than Do to 20
D. 4Ø IF A / B <> C + D ^ 2 THEN 235 If A Divided by B is not equal to C plus D to the Power of 2 than Do to 235

3. Draw flowchart symbols for each of the statements in problem 2.

If-then Symbol or Decision Symbol

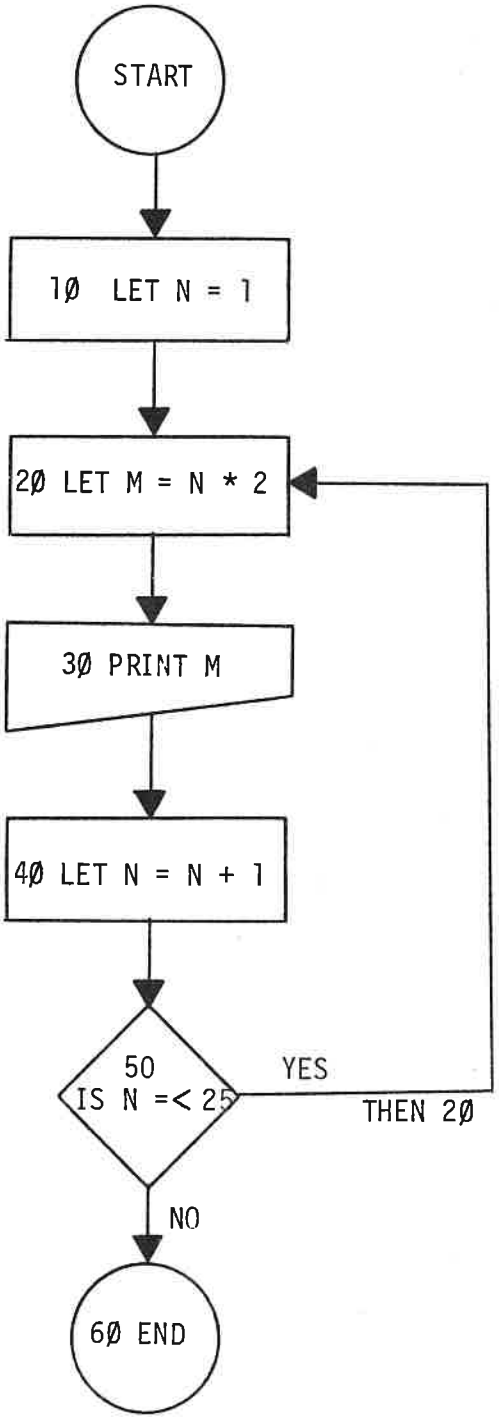


4. Here is a portion of a program and five versions of line 13Ø. For each, decide if the condition is TRUE or FALSE and give the line number of the statement to be executed next:

```
1ØØ  LET A = 16
11Ø  LET B = 24
12Ø  LET C = 48
13Ø  ----------
14Ø  ----------
```

Statement 13Ø	Condition Is	Line Number
A. 13Ø IF C > A THEN 6Ø	TRUE	6Ø
B. 13Ø IF A = C THEN 6Ø	FALSE	14Ø
C. 13Ø IF A / 2 8 = C / 2 B THEN 6Ø	TRUE	6Ø
D. 13Ø IF B <> C 3 - 2 A THEN 6Ø	TRUE	6Ø
E. 13Ø IF C <> 2 3 * 2 A THEN 6Ø	TRUE	6Ø

5. Study the flowchart below and then answer the questions:



25

A. What is the output for this program?
B. Locate the decision box. If the box were replaced by another containing IS N < 25? what would the effect be on the output? 24
C. Write a program from this flowchart and run it. Compare the printout with your results in A.

6. Write a program that will print in columns the square and cube of the even numbers from 18 to 36. Print headings on the columns.

7. Modify the program in problem 6 to include INPUT statements so the user can select starting and ending values.

8. Assume you have been offered a very risky job for one month. The employer has given the choice of two salary plans:

   A. $2Ø.ØØ per week for one month, or
   B. $.Ø1 for the first day with your daily salary doubling each day you are on the job for the thirty day month

   Draw a flowchart to help you choose the best salary plan. Then write a program from your flowchart and run it on the microcomputer. How much would you make for a month under plan B? What is the salary for the 30th day? If the month had 31 days, how much money would you make on the 31st day under plan B?

9. Here is a printout from a BASIC program execution. Study the sample run. Then draw a flowchart for a program to accomplish this same printout. Next, write a program from your flowchart and execute it.

   Problem

   This problem has the following options:

   A. Calculate the square of any positive number entered
   B. Calculate the cube of any negative number entered
   C. Terminate the execution if a zero is entered

   Sample Run

```
ENTER A NUMBER OF YOUR CHOICE, POSITIVE, NEGATIVE, OR ZERO
? 25

THE SQUARE OF 25 IS 625

ENTER A NUMBER OF YOUR CHOICE, POSITIVE, NEGATIVE, OR ZERO
? -369

THE CUBE OF -369 IF -5Ø243Ø9

ENTER A NUMBER OF YOUR CHOICE, POSITIVE, NEGATIVE, OR ZERO
? Ø

END OF PROGRAM
```

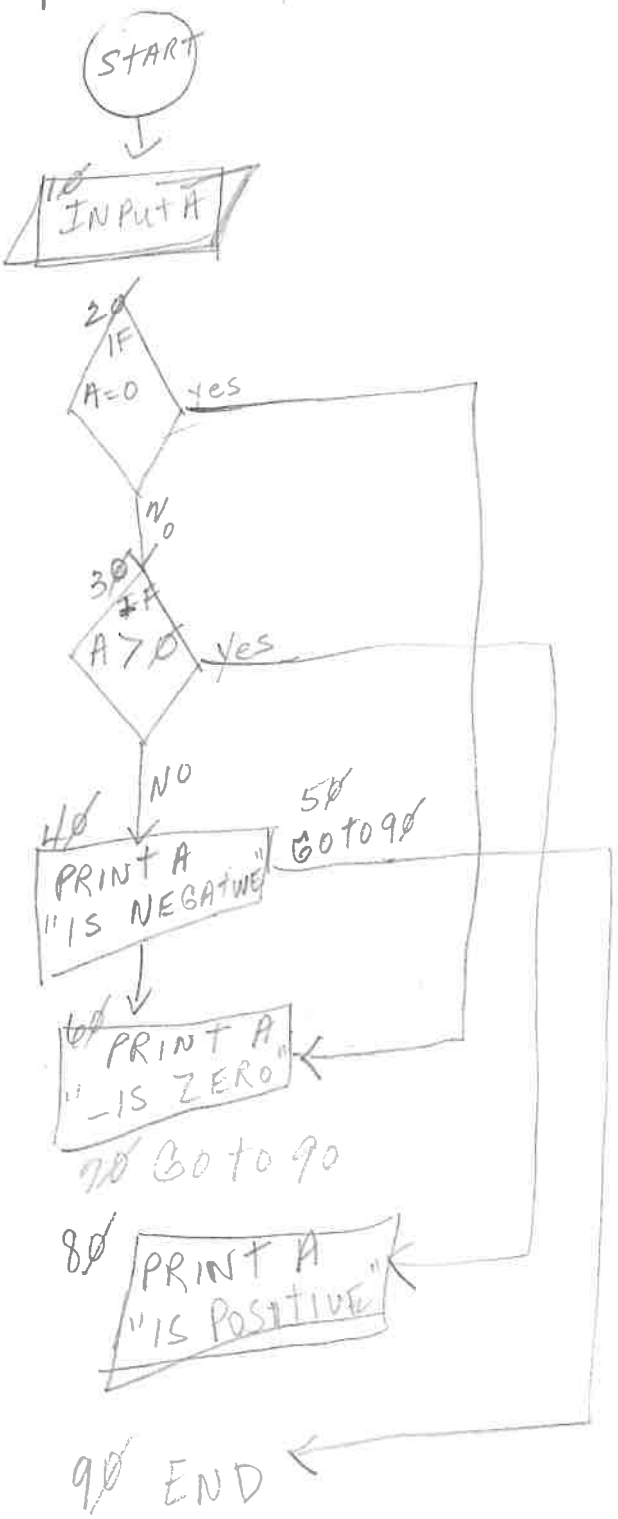
1Ø. Study the following program:

```
1Ø  INPUT A
2Ø  IF A = Ø THEN 6Ø
3Ø  IF A > Ø THEN 8Ø
4Ø  PRINT A;" IS NEGATIVE"
5Ø  GO TO 9Ø
```

```
60  PRINT A;" IS ZERO"
70  GO TO 90
80  PRINT A;" IS POSITIVE"
90  END
```

A. Draw the flowchart for this program
B. Rewrite the program using the second form of the IF-THEN statement
C. Draw the flowchart for your program in part B

A.



2 - 4 FOR-NEXT Statements

The FOR and NEXT statements shorten and simplify the writing of loops. Look at these two programs. One uses an IF-THEN statement; the other uses a FOR-NEXT loop. Both programs do the same thing.

IF-THEN

```
100  LET N = 1
120  PRINT N, N ^ 2
130  LET N = N + 1
140  IF N < 10 THEN 120
150  END
```

FOR-NEXT

```
100  FOR N = 1 TO 10
110  PRINT N, N ^ 2
120  NEXT N
130  END
```

These two programs both have loops which:

Start at N equal to 1

Print N and its square

Are executed 10 times

Loops have the following components or actions:

Initial value assigned to a counter

→ Statements to be repeated are executed

The counter is incremented

The counter is tested for a final value

— If the counter is not greater than the final value, the loop is repeated.

Otherwise the loop is terminated → END.

FOR-NEXT statements automatically do these things:

FOR -- sets the initial value for the counting variable and increments the counting variable each time the loop is executed.

NEXT -- tests for the final value of the counting variable and either sends control back to the FOR statement or terminates the loop.

general form:

```
LN  FOR sv = i TO f

    statements to be repeated

LN  NEXT sv
```

where:

sv -- is the counting variable. The same variable must be used in both the FOR and NEXT statements.

i -- is the initial value assigned to the counting variable. This may be a number, a variable, or an expression.

f -- is the final value for the counting variable. This may be a number, a variable, or an expression.

HOW THE FOR-NEXT STATEMENTS WORK

Examine the sample program and read the description of what the program is doing.

sample:

```
1Ø  FOR A = 1 TO 3
2Ø  PRINT "STAMP YOUR FOOT"
3Ø  NEXT A
4Ø  END
```

description:

1 - Start. 1 is assigned to the counting variable.
STAMP YOUR FOOT is printed.
TEST - (is the value of the counting variable less than the final value?) i.e., is 1 less than 3?
YES - go back to the FOR statement.

2 - Increment the counting variable.
A is now 2.
STAMP YOUR FOOT is printed.
TEST - is 2 less than 3?
YES - go back to the FOR statement.

3 - Increment the counting variable.
A is now 3.

STAMP YOUR FOOT is printed.
TEST - is 3 less than 3?
NO - the loop is now terminated.

The loop is executed three times. The counting variable A is assigned the number (1, 2, 3), one for each time the FOR statement is executed. The test "is A less than 3" is being made each time the next statement is reached. If the condition is TRUE, a GO TO the FOR statement occurs. If the condition is FALSE, the statement following NEXT is executed and the loop terminates.

FOR statements can start with numbers other than 1:

example:

```
10  FOR M = 10 TO 12
20  PRINT M
30  NEXT M
40  END
```

The computer executes the loop three times. The values assigned to M and printed are:

example:

```
10
11
12
```

STEPPING

The FOR statement may increment by values other than 1. For example, we can print the even numbers from 1 to 10 by:

example:

```
10  FOR X = 2 TO 10 STEP 2
20  PRINT X
30  NEXT X
40  END
```

The numbers the counting variable uses are (2, 4, 6, 8, 1Ø). STEP tells the microcomputer what number to add to the counting variable each time the FOR statement is repeated.

Counting backwards can be accomplished by using a STEP -1. Using a negative step value means the test being made by the NEXT statement will change to "is greater than." The initial assignment value should be larger than the final value for the loop to operate properly.

example:

```
1Ø  FOR I = 5 TO 2 STEP -1
2Ø  PRINT I
3Ø  PRINT I
4Ø  END
```

This time the values assigned to the counting variables are (5, 4, 3, 2). The STEP value can be a number, variable, or an expression.

NESTED FOR NEXT LOOPS

More than one FOR-NEXT loop may be used in a program. They may also be placed inside one another (nested).

example:

range of loop

```
1Ø  FOR A = 1 TO 3
2Ø  PRINT "THIS STATEMENT IS IN THE RANGE OF LOOP A"
3Ø  FOR B = 1 TO 5
4Ø  PRINT "THIS STATEMENT IS IN THE RANGE OF LOOPS A AND B"
5Ø  FOR C = 7 TO 11 STEP 2
6Ø  PRINT "THIS STATEMENT IS IN THE RANGE OF LOOPS A, B,
    AND C"
7Ø  NEXT C
8Ø  NEXT B
9Ø  NEXT A
1ØØ  END
```

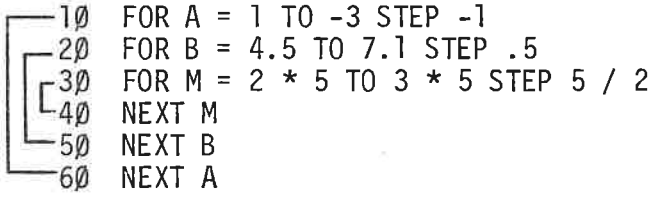
A loop's range is the group of statements in a FOR-NEXT loop.

The ranges of nested loops may not overlap. The range can easily be determined by drawing a bracket from the FOR statement to its NEXT statement.

If the brackets for each loop's range do not cross each other, the nesting has been done properly as in the following example:

example:

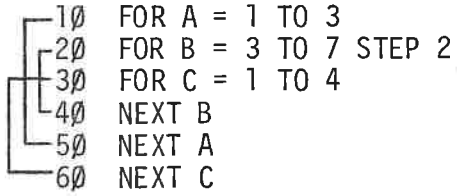
correctly nested loops



Incorrectly nested loops will have crossed brackets as in the following example:

example:

incorrectly nested loops



Exercise 2 - 4

1. The FOR and NEXT statements cause one or more instructions to be:

2. Give the output for each of these programs:

   A.
```
5  LET B = Ø
1Ø  FOR A = 1 TO 5
2Ø  LET B = B + A
3Ø  PRINT B
4Ø  NEXT A
5Ø  END
```

   B.
```
1Ø  FOR W = 1Ø TO 15 STEP 8
2Ø  PRINT W
3Ø  NEXT W
4Ø  END
```

   C.
```
2Ø  FOR W = 1 TO 2 STEP .2
3Ø  PRINT W
4Ø  NEXT W
5Ø  PRINT W,W;W
6Ø  END
```

   D.
```
1Ø  FOR X = 1Ø TO 1 STEP -1.5
2Ø  PRINT X
3Ø  LET X = X - 1
4Ø  NEXT X
5Ø  PRINT X
6Ø  END
```

3. When you are stepping with a positive number using a FOR statement, the larger number is the:

   A. Initial value
   B. Final value

4. The FOR statement determines a set of values for the counting variable. Determine the set of values for each of these FOR statements:

   A. FOR W = 1 TO 5
   B. FOR D = 4 TO 18 STEP 3.5
   C. FOR P = 5 TO -3 STEP -2
   D. FOR A = 2 TO 4 STEP .5

5. Write FOR statements which use the following sets of numbers for the counting variables:

   A. (Ø, 3, 6, 9, 12)
   B. (1Ø, 1ØØ, 1ØØØ, 1ØØØØ)
   C. (2.5, 3.Ø, 3.5, 4.Ø)
   D. (5, -2, -9, -16, -23)

6. Find all the errors in the following program:

```
1Ø  INPUT A
2Ø  IF A = Ø THEN 35
3Ø  FOR I = 1 TO A
4Ø  PRINT I * A
5Ø  NEXT A
6Ø  GO TO 4Ø
7Ø  END
```

7. Both the GO TO and the IF-THEN statements may cause INFINITE LOOPS. Can a FOR-NEXT loop be written in such a way as to cause an INFINITE LOOP?

8. Print the multiplication table for sevens from one to twelve.

9. Given an hourly wage, calculate the yearly income and total income for the next 1Ø years. Figure an 8% raise in salary each year. Use forty hours per week and fifty-two weeks in a year.

1Ø. Find how much interest will be received on $125 over 12 years. Figure 8.25% simple interest, no compounding.

11. Re-write the program in 12, but compound the interest yearly. Print out how much interest is received each year.

12. Write a program that will accept one to ten numbers to be entered and print the sum and average of the numbers.

Review - Chapter II

In this chapter the statements which allow the repeated execution of other statements were introduced.

DEFINITIONS

LOOP -- A group of statements which are repeated in a program.

LOOPING -- The execution of a loop.

```
          10  LET C = 1
        ┌ 20  PRINT C
loop:   │ 30  LET C = C + 1
        └ 40  IF C < 10 THEN 20
          50  END
```

Statements 20, 30, and 40 form a loop. These statements are executed 10 times in this example.

BRANCHING -- Branching is the process of "jumping over" one or more statements without executing them. In the above example, statement 40 causes a branch to statement 20, jumping over statement 30.

CONTROLLED LOOP -- A loop which is executed a limited number of times is called a controlled loop because it is "controlled" by a counter and a decision statement. Controlled loops have the following components or actions:

A. A counter is set to a starting value
B. One or more statements to be repeated
C. The counter is incremented
D. A decision is made to either continue or exit the loop

CONCEPTS AND PROGRAMMING TECHNIQUES

REPLACEMENT -- The assignment statement "LET" was re-defined as a replacement statement.

```
10  LET A = 10
```

Replacement means: The value assigned and the variable appearing to the left of the assignment operator is "REPLACED" by the value appearing to the right of the assignment operator.

In the above example the previous value assigned to the variable A is replaced by 10.

COUNTING -- By using the concept of REPLACEMENT, the LET statement can be used to count.

```
1Ø  LET A = 1
2Ø  LET A = A + 1
```

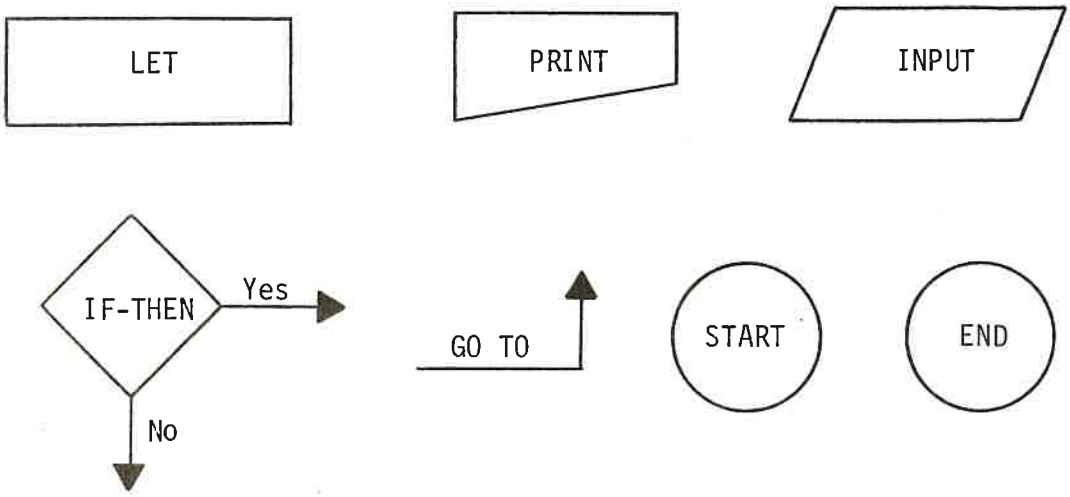
Statement 2Ø causes 1 to be added to the value assigned to A each time the statement is executed. The new value replaces the old value.

```
3Ø  LET A = A + 5
```

Any increment value, positive or negative, can be used. This example would allow us to count by 5's.

FLOWCHARTING -- Flowcharting is the process of "drawing" a picture of the solution to a problem using special symbols. The symbols used in this manual were chosen because they are easy to draw using just a pencil and paper. Each symbol corresponds to a BASIC statement so that when finished with a flowchart, a student may write a BASIC program from it with ease.

FLOWCHART SYMBOLS

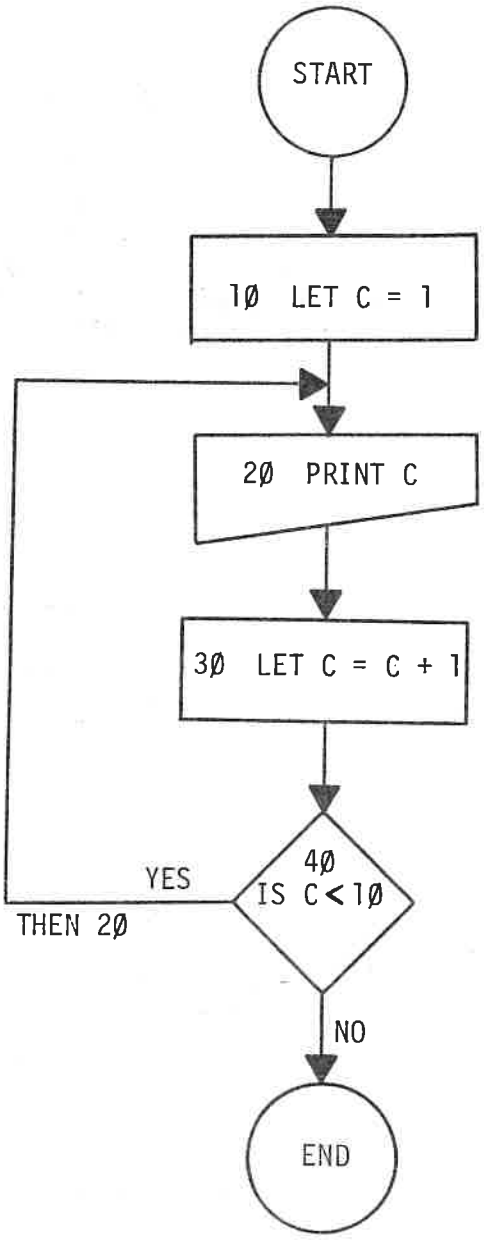


In addition to these symbols, arrows are used to connect boxes indicating the order in which statements are executed.

Flowcharts are valuable tools for solving complex problems and for detecting FLAWS and INCORRECT solutions.

TRACING

Tracing is a pencil and paper tool for reading and analyzing flowcharts. Each time a variable is to be assigned a value, or information is to be printed, the student records this on a lined sheet of paper.



C	OUTPUT
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
1Ø	

The variable(s) are listed with the values assigned, forming columns. Each time the value changes, the old value is crossed out with the new value placed in the column. Information to be printed is listed, as it should

appear on the right hand side of the paper. By carefully tracing flowcharts logic errors and incorrect solutions can be found before the program is written.

BASIC STATEMENTS

GO TO m -- The GO TO statement causes an unconditional transfer of control to the line number "m" when executed.

IF (R) THEN m -- The IF-THEN statement evaluates a RELATIONAL expression (R) and assigns a value of TRUE or FALSE to the expression. If the value assigned is TRUE, the THEN portion of the statement is executed which transfers program control to line number "m". If the value assigned is FALSE, the THEN portion of the statement is ignored. The next statement in sequence is then executed.

RELATIONAL EXPRESSIONS are written as:

Numerical Expression	Relational Operator	Numerical Expression

The two numerical expressions are first evaluated. Their values are then compared. The relational operator determines the kind of comparison to be made.

Relational Operators	Decision to be Made
=	is equal to
>	is greater than
<	is less than
> =	is greater than or equal to
< =	is less than or equal to
< >	is not equal to

IF (R) THEN (instruction) -- The second form of the IF-THEN can be used to eliminate steps in programming. If the relational expression (R) is true, the INSTRUCTION is performed and then the next statement in the program is executed. If the relational expression is false, the instruction is ignored and the next statement in the program is executed.

FOR-NEXT -- The FOR-NEXT statements are used as a pair to shorten and simplify the writing of loops. FOR I = i to f step s

The FOR statement causes an initial value i to be assigned to the counting variable. Each time the statement is repeated, the step value

s is added to the counting variable. The final value f is the largest value the counting variable may have. When the counting variable exceeds the final value, the loop is terminated.

If the step value is negative, the final value is the smallest value the counter variable may have.

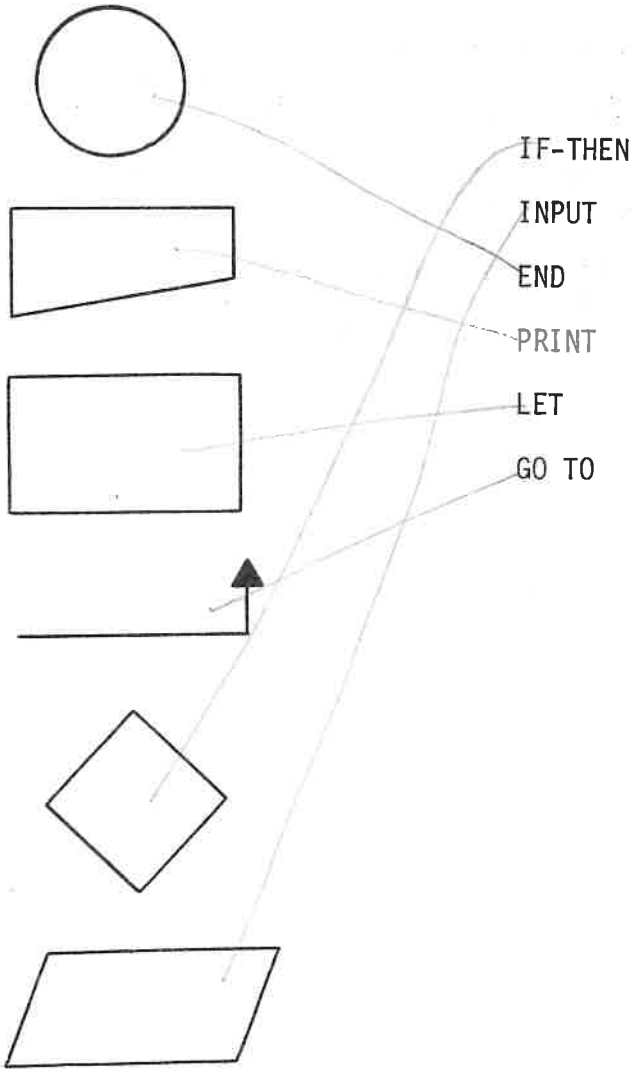
The NEXT statement tests the counting value against the final value and either sends control back to the FOR statement or terminates the loop. The type of comparison made depends on the sign of the STEP value. A positive step value causes a test of "is less than" to be made, while a negative step value causes a test of "is greater than" to be made.

Review Quiz - Chapter II

1. Define the following terms:

   A. Loop
   B. Branch
   C. Infinite loop
   D. Controlled loop

2. Match each flowchart symbol on the left with its corresponding BASIC statement on the right:



3. Write a program that will make a right triangle. The outline of the triangle should be *'s and the inside of the triangle should be T's.

4. Draw a flowchart for problem 4.

5. Draw a flowchart for this program:

```
1Ø  FOR X = 1 TO 3
2Ø  FOR Y = 3 TO 5
3Ø  PRINT "*";
4Ø  IF X + Y = 6 THEN 7Ø
5Ø  NEXT Y
6Ø  PRINT " ";
7Ø  PRINT "666"
8Ø  NEXT X
9Ø  END
```

6. Trace the flowchart in problem 6. Does it have a loop? If yes, how many? Does it have any infinite loops?

7. Write a program that will make a calendar for any month given what day the month starts with and how many days are in the month. For the starting day use 1 for Sunday, 2 for Monday, etc.

8. List each of the relational operators that may appear in an IF-THEN statement and its meaning.

9. Using paper .ØØ3 inches thick, make a stack. First add two sheets, then four sheets, then eight, etc. How many additions are needed to make a stack one mile high? One mile is 6336Ø inches.

Chapter III

FUNCTIONS

Chapter III introduces the most frequently used BASIC functions for performing calculations: these are the greatest integer and the random number. Functions may appear in the LET, PRINT, IF-THEN, and FOR statements. Being lengthy and difficult to write, functions were developed to simplify performing calculations in programs. When executed, the result of their computation is substituted for the function itself in the statement. These two functions will be used in the graphics section also included in this chapter.

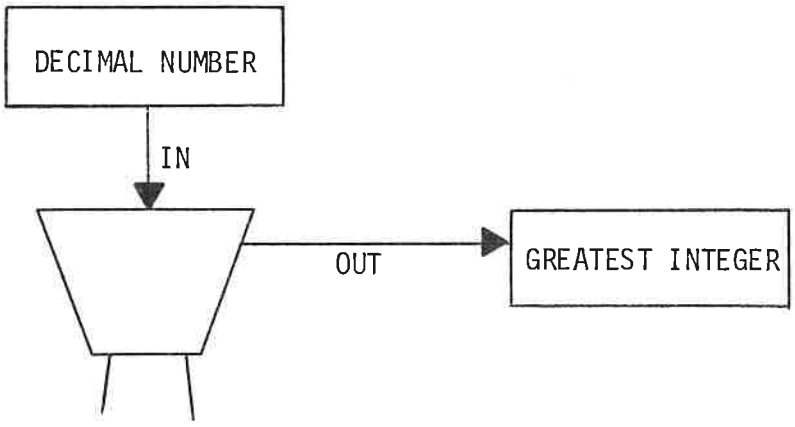
3 - 1 Greatest Integer Function

general form: INT(e)

where: e may be a number, variable, or expression.

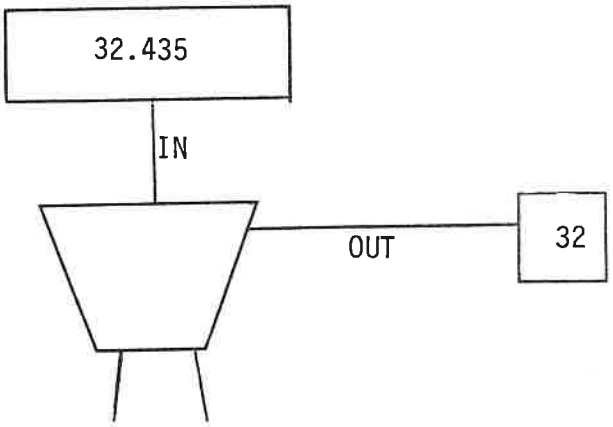
purpose: The INT function returns the nearest integer to the number entered which is not greater than the original value.

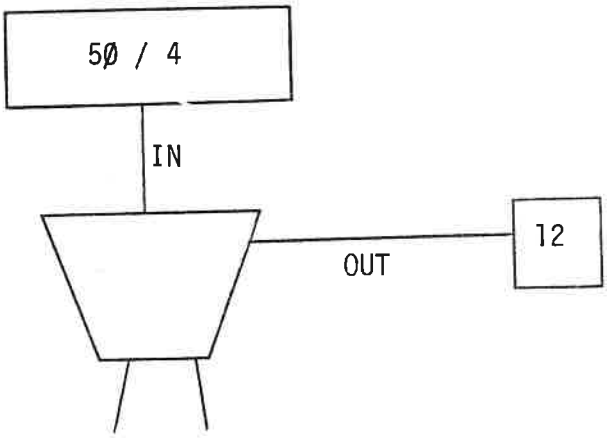
The greatest integer function may be thought of as a number machine. Each time you put a number in you get the nearest integer not greater than the original value out. The INT function is used most frequently with decimal numbers.

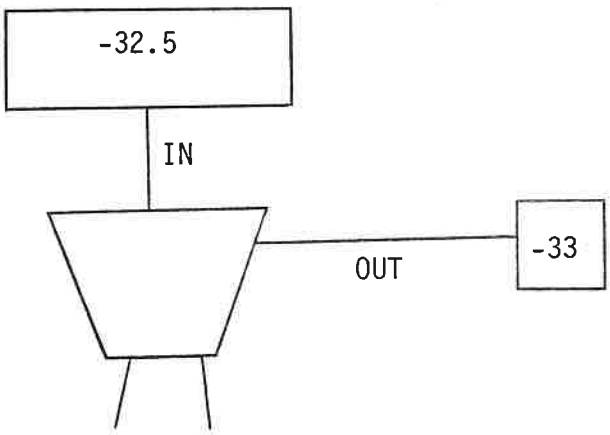


examples:

```
10  C = INT (32.325)
20  D = INT (50 / 4)
30  C2 = INT (-32.5)
```







Compare each of the numbers entered with the value returned. Draw a number line and locate the numbers. Can you state a general rule for INT and the number line?

problem 1: A number is "even" if the number divided by 2 results in an integer. In BASIC, we can say this as:

```
IS (N / 2) = INT (N / 2)  ?
```

For example, testing 1Ø for an even number results in:

```
IS (1Ø / 2) = INT (1Ø / 2)  ?
IS 5        = 5             ?
```

Since they are equal, 1Ø is an even number.

Testing 11 for being even:

```
IS (11 / 2) = INT (11 / 2)  ?
IS 5.5      = 5             ?
```

No, this time they are not equal. 11 is an odd number.

Write a BASIC program to tell if a number entered is EVEN or ODD.

program 1:

```
1Ø  PRINT "ENTER A NUMBER AND I WILL TELL YOU - EVEN OR ODD"
2Ø  PRINT
3Ø  INPUT N
4Ø  IF N / 2 = INT (N / 2) THEN 7Ø
5Ø  PRINT "ODD"
6Ø  GO TO 8Ø
7Ø  PRINT "EVEN"
8Ø  END
```

problem 2: A number is a factor of another number if it will divide into the second number without a remainder. This is similar to the EVEN - ODD problem.

In BASIC the INT function can be used to find factors of numbers.

Write a program to determine if 3 is a factor of 2Ø.

program 2:

```
1Ø  LET N = 2Ø
2Ø  LET D = 3
3Ø  IF N / D < > INT (N / D) THEN 6Ø
4Ø  PRINT D;" IS A FACTOR OF ";N
```

```
50  GO TO 70
60  PRINT D;" IS NOT A FACTOR OF ";N
70  END
```

problem 3: We can easily expand program 4 to find all the factors of a number. We will count from 1 to the number and test each counting number for being a factor.

program 3:

```
10  PRINT "ENTER A NUMBER GREATER THAN 0"
20  INPUT N
30  FOR X = 1 TO N / 2
40  IF N / X <> INT (N / X) THEN 60
50  PRINT X
60  NEXT X
70  END
```

problem 4: Write a program to print the greatest common divisor of two numbers 28 and 49.

program 4:

```
10  FOR I = 1 TO 28
20  IF 28 / I <> INT (28 / I) THEN 50
30  IF 49 / I <> INT (49 / I) THEN 50
40  LET G = I
50  NEXT I
60  PRINT "THE GREATEST COMMON DIVISOR"
70  PRINT "OF 49 AND 28 IS"; G
80  END
```

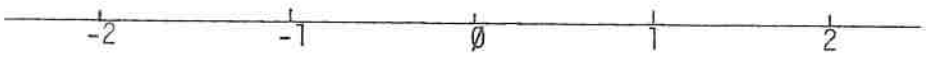
Exercise 3 - 1

1. What is the purpose of the INT function?

2. What will be printed by each of the following statements?

   A. 10 PRINT INT (100)
   B. 20 PRINT INT (-100)
   C. 30 PRINT INT (100.1)
   D. 40 PRINT INT (100.9)
   E. 50 PRINT INT (-100.9)

3. Plot each of the following values on the number line:



   A. INT (.6)
   B. INT (1.1)
   C. INT (1.8)
   D. INT (-1.3)
   E. INT (-.4)

4. State a rule for the location of a number and the location of its INT value on a number line.

5. Write statements to find the INT value for each of the following:

   A. 1ØØ * 5.73
   B. 17.983 / 4
   C. 2.5 * INT (6.7)
   D. 6 ^ 3

6. Write a program that will determine if nine is a factor of a given number. (Use sample program 1 if you need help.)

7. With the help of sample program 2 change or add statements to allow the program to determine if any number entered is a factor of any other number entered.

8. Write a program that will find the factors of a given number.

9. Write a program that will print all the prime numbers between 1ØØ and 3ØØ.

10. Write a program that will find the greatest common divisor of any two numbers entered. Also print all the common divisors. If you need help, use sample program 4.

11. The INT function is used to help "round" numbers. The following program shows how the number 12.3456789 could be rounded to hundredths place:

```
1Ø  LET A = 12.3456789
2Ø  LET A = A + .ØØ5
3Ø  LET A = A * 1ØØ
4Ø  LET A = INT(A)
5Ø  LET A = A / 1ØØ
6Ø  PRINT A
7Ø  END
```

   Rewrite this program to allow any number to be rounded to any place.

3 - 2 Random Numbers

THE COIN TOSS SIMULATION

Tossing a coin results in either a HEAD or a TAIL. If you tossed a coin several times, you should notice that about half the time you get a HEAD and half the time you get a TAIL. If you tossed the coin several hundred times, the number of heads and tails would be closer to the same. This kind of EVENT is called a Random Chance Event because we cannot predict exactly the outcome.

In this section we will instruct the computer to simulate the random event of tossing a coin. To start, we will need to know how to instruct the computer to generate RANDOM NUMBERS. The numbers are called RANDOM because the likelihood of getting one number is about the same as getting another.

RANDOM NUMBER FUNCTION

general form: RND(x)

where: If x is positive, a different random number is generated each time the function is executed.

If x is negative, the same random number is generated for the specific negative number.

If x is zero, the last generated random number is returned.

This manual will be using a positive one for the argument x.

purpose: Each time the RND(1) is executed, a number greater than or equal to zero and less than one is generated.

example:

```
10  FOR I = 1 TO 4
20  X = RND(1)
30  PRINT X
40  NEXT I
50  END
```

This program prints:

```
.973136996
.1Ø3117626
.Ø177148333
.779343355
```

The next time the program is run the numbers would be different. What do numbers like these have in common with tossing a coin? First, the likelihood of any of the numbers being selected is the same. If the list of numbers are divided into two parts, one list being the numbers less than .5 and the other list being the numbers greater than or equal to .5, then for any practical use the lists will be considered as being the same. If HEAD is assigned to the numbers less than .5 and a TAIL to any other number, this results in the same conditions as a coin flip, half heads and half tails.

We are now ready to instruct the computer to "simulate tossing a coin."

program:

```
2Ø  LET A = RND(1)
3Ø  IF A < .5 THEN 6Ø
4Ø  PRINT A, "TAIL"
5Ø  GO TO 7Ø
6Ø  PRINT A, "HEAD"
7Ø  END
```

running the program:

```
.115622Ø79                    HEAD
```

Since the number printed was less than .5 it was assigned a HEAD. To verify that about half the time you run this program you will get a HEAD printed, you can run a second program. In this program the number of times HEAD and TAIL are selected will be counted.

program:

```
2Ø  FOR X = 1 TO 1ØØØ
3Ø  LET A = RND(1)
4Ø  IF A < .5 THEN 7Ø
```

```
5Ø  LET T = T + 1
6Ø  GO TO 8Ø
7Ø  LET H = H + 1
8Ø  NEXT X
9Ø  PRINT "HEADS = ";H,"TAILS = ";T
1ØØ  END
```

Statements 5Ø and 7Ø count the number of times a tail and head condition are generated. Statement 9Ø prints the results of the counting. (Note: This program will take about 15 seconds to run.)

RANDOM AND INT

By combining the Greatest Integer Function and RND, the coin toss can be made easier to understand.

program:

```
2Ø  LET A = RND(1)
3Ø  LET B = 2 * A
4Ø  LET C = INT(B)
5Ø  PRINT A, B, C
6Ø  END
```

Two sample runs:

```
.Ø955942195          .191188439          Ø
.736816379          1.47363276           1
```

The sample runs demonstrate how a decimal number can be changed to one of two single digit integers, Ø or 1.

Let's review what has been discussed. By multiplying a random number by 2 and taking the integer part, about half of the possible numbers generated would be changed to a zero and about half would be changed to a one.

The coin toss can now be simulated by the following program:

program:

```
3Ø  LET B = INT(2 * RND(1))
4Ø  IF B < 1 THEN 7Ø
5Ø  PRINT "TAIL"
6Ø  GO TO 8Ø
7Ø  PRINT "HEAD"
8Ø  END
```

ROLLING DICE

Rolling dice is similar to tossing a coin. The chance of any one of the six sides appearing is the same (if the dice are fair). To "simulate" the rolling of a single dice we could divide the possible random numbers list into six equal parts by:

1 Selecting a random number
2 Multiplying the random number by 6
3 Adding 1 to the result
4 Taking the integer part of the result

The numbers 1, 2, 3, 4, 5, 6 each have an equal chance of being selected. The statement: 2Ø LET A = INT(6 * RND(1) + 1) will accomplish this task.

Statement 2Ø multiplies the random number by 6 changing the possible number selected to being greater than or equal to Ø and less than 6.

Next, adding "one" changes the number selected again to be greater than or equal to 1 and less than 7.

Finally, taking the INT value of the result gives one of the single digits 1, 2, 3, 4, 5, or 6.

Given a large sample, the number of times each of the single digits appear is so close to the same, we assume they are equal.

Next to proceed with the simulation, PRINT statements are created which show the face of the dice.

The empty dice can be printed by the following:

```
100  PRINT "- - - - - - -"
110  PRINT "-           -"
120  PRINT "-           -"
130  PRINT "-           -"
140  PRINT "- - - - - - -"
```

Putting it all together the following dice rolling simulation program results:

```
10  PRINT "- - - - - - -"
20  LET A = INT(6 * RND(1) + 1)
30  IF A = 1 THEN 600
40  IF A = 2 THEN 500
50  IF A = 3 THEN 400
60  IF A = 4 THEN 300
70  IF A = 5 THEN 200
80  PRINT "- * * * -"
90  PRINT "-       -"
100  PRINT "- * * * -"
110  GO TO 700
200  PRINT "-  * *  -"
210  PRINT "-   *   -"
220  PRINT "-  * *  -"
230  GO TO 700
300  PRINT "-  * *  -"
310  PRINT "-       -"
320  PRINT "-  * *  -"
330  GO TO 700
400  PRINT "- *     -"
410  PRINT "-   *   -"
420  PRINT "-     * -"
430  GO TO 700
500  PRINT "-      *-"
510  PRINT "-       -"
520  PRINT "-*      -"
530  GO TO 700
```

```
600  PRINT "-           -"
610  PRINT "-     *     -"
620  PRINT "-           -"
700  PRINT "- - - - - - -"
710  END
```

Exercise 3 - 2

---

1. Describe what a "Random Chance Event" is.

2. What range of numbers are generated by RND(1)?

3. What happens to the range of numbers if we multiply RND(1) by 7?

4. What will the value of INT(RND(1)) always be? Why?

5. Will the value of RND(5) always be the same? If yes, what is its value?

6. Will the value of RND(-5) always be the same? If yes, what is its value?

7. Given the program:

```
10  LET A = RND(-5)
20  LET B = RND(5)
30  END
```

   What is the value of B? Will it always be the same for this program?

8. What values could be assigned to variable C by the following statement?

```
40  LET C = INT(2 * RND(1) + 10)
```

9. Write a BASIC program to select a random number from the possible numbers (-2, -1, 0, 1, 2).

10. Write a statement that would generate random numbers between .25 and .5.

3 - 3 Low Resolution Graphics

The microcomputer is capable of changing the printing screen to a graphic screen and back. Four programming statements are needed to do this. They are: GR, COLOR, PLOT, and TEXT.

general form: GR with no argument

purpose: This statement clears the screen to black and sets low resolution graphics. Low resolution graphics turn the the screen into a grid of 40 points by 40 points with four lines at the bottom of the screen for normal printing.

Before any points can be plotted on the screen, the color to be used must be set. The colors that can be used and their corresponding equivalent number is:

0 - black	6 - medium blue	12 - green
1 - magenta	7 - light blue	13 - yellow
2 - dark blue	8 - brown	14 - aqua
3 - purple	9 - orange	15 - white
4 - dark green	10 - grey	
5 - grey	11 - pink	

general form: COLOR = x

where: x is one of the above numbers. COLOR is set to zero or black after the statement GR. x can be any numeric expression.

purpose: When executed the color is set so plotting of points can take place.

The plotting of points can now begin with the use of the following statement.

general form: PLOT X,Y

where: X is the X-coordinate and Y the Y-coordinate. The range of X and Y must be from Ø to 39. Y can also have the values of 4Ø to 47, but odd things can happen when they are used. X and Y can be any numeric expression.

If the GR statement is not used before the statement PLOT, this will cause strange effects. Point Ø,Ø is in the upper left corner of the screen, point 39,39 is in the lower right corner of the screen.

example:

```
1Ø  GR
2Ø  COLOR = 6
3Ø  PLOT 1Ø,1Ø
4Ø  END
```

Enter and run the above program. The program should plot a blue square. Now use the LIST command to list the program. What happens? The program should only list on the bottom four lines of the screen and the blue square will stay where it is.

To go back to the printing screen or otherwise called the TEXT screen, the TEXT command has to be used.

general form: TEXT with no argument

where: This command can be used with or without a line number.

purpose: To go from graphics mode to text mode.

Enter TEXT at this time without a line number. The screen will change back to text and be filled with a lot of characters. Now, type LIST to see a listing of the program. On the following page is a sample program that uses TEXT as a statement.

example:

```
10  GR
20  COLOR = 6
30  PLOT 10,10
40  GET A$
50  TEXT
60  END
```

A program that will randomly select any of the fifteen colors used in low resolution graphics is listed below:

example:

```
10  GR
20  LET X = INT(15 * RND(1) + 1)
30  COLOR = X
40  PLOT 20,20
50  GET A$
60  TEXT
70  END
```

Exercise 3 - 3

---

1. Why must GR be used before PLOT?

2. Why must COLOR be used before PLOT?

3. Write a program to plot a green point at 10,10; a blue point at 10,30; a brown point at 30,30; and a yellow point at 30,10.

4. Write a program that will randomly plot 10 green points anywhere on the screen.

5. Write a program that will randomly generate colors and points and plot them on the screen.

6. Re-write the program in problem 5 so that the X-coordinate and Y-coordinate can have only variables from 10 to 25.

7. Write a program that makes a border of orange around the screen.

3 - 4 Special Plotting Functions

There are two special functions to help plot vertical and horizontal lines. The first one is HLIN and is used in plotting horizontal lines.

general form: HLIN X,Z AT Y

where: The points plotted will be from point X,Y to point Z,Y.

purpose: To quickly plot horizontal lines with one statement.

sample:

```
10  GR
20  COLOR = 3
30  HLIN 5,15 AT 20
40  END
```

The points 5,20; 6,20; 7,20; 8,20; 9,20; 10,20; 11,20; 12,20; 13,20; 14,20; and 15,20 will be plotted.

The second function is VLIN and is used to plot vertical lines.

general form: VLIN Y,Z AT X

where: The points plotted will be from point X,Y to point X,Z.

purpose: To quickly plot vertical lines with one statement.

sample:

```
10  GR
20  COLOR = 1
30  VLIN 5,15 AT 10
40  END
```

The points 10,5; 10,6; 10,7; 10,8; 10,9; 10,10; 10,11; 10,12; 10,13; 10,14; 10,15 will be plotted.

Exercise 3 - 4

1. Re-write the program in exercise 3-3, problem 7, using HLIN and VLIN.

2. Write a program using HLIN or VLIN and make the screen blue.

3. Plot horizontal lines on the screen using all the different colors. Make a black line between each of the colored lines.

4. What must be changed in problem 3 to plot vertical lines instead of horizontal lines?

5. Randomly plot different colored horizontal lines where the Y-coordinates are from 15 to 3Ø and then plot vertical lines where the X-coordinates are from 15 to 3Ø.

6. Re-write problem 5 so first a horizontal line is plotted and then a vertical line for the same X and Y coordinates.

7. Re-write problem 1 so it will keep making smaller and smaller borders until the screen is full.

3 - 5 Animation

With the capability of graphics display it is possible to make graphics figures appear as they are moving. This concept is called animation. The first example shows how a solid figure could be plotted.

example:

```
100  GR
110  COLOR = 6
120  FOR I  = 1 TO 4
130  HLIN 18,21 AT I
140  NEXT I
150  END
```

When you run the above program a blue square will be generated at the center top of the screen. The square is generated with the top left-hand corner at point (18,1), top right-hand corner at point (21,1), bottom left-hand corner at point (18,4), and the bottom right-hand corner at point (21,4).

Study the next example that makes the square animate down the screen.

example:

```
90  TEXT
100  GR
110  COLOR = 6
120  FOR I = 1 TO 4
130  HLIN 18,21 AT I
140  NEXT I
150  FOR X = 1 TO 36
160  COLOR = 0
170  LET A = A + 1
180  HLIN 18,21 AT A
190  COLOR = 6
200  LET B = A + 4
210  HLIN 18,21 AT B
220  NEXT X
230  END
```

When you run the above program the blue square will animate down the screen.

Line 16Ø sets the color to black. Lines 17Ø and 18Ø make the top one-fourth of the square black. Lines 19Ø through 2ØØ adds another blue line to the bottom of the square. This process makes the animation. The loop in line 15Ø causes the process to stop after 36 additions and subtractions from the square.

With the addition of the following statement the color of the square would be random as the square moves down the screen.

```
19Ø  COLOR = INT(15 * RND(1) + 1)
```

Exercise 3 - 5

1. Describe the result of the following step:

   ```
   1ØØ  HLIN 1Ø,2Ø AT 1Ø
   ```

2. Describe the result of the following step:

   ```
   1ØØ  VLIN 5,6 AT 2Ø
   ```

3. Describe the graphics that would be generated by the following program:

   ```
   1ØØ  GR
   11Ø  COLOR = 3
   12Ø  FOR I = 1 TO 2
   13Ø  HLIN 1Ø,2Ø AT I
   14Ø  NEXT I
   15Ø  END
   ```

4. Describe what is meant by animation.

5. Write a program that will generate a moving square across the horizontal part of the screen. Start with the coordinates (1,18), (4,18), (1,21), (4,21).

Review - Chapter III

In Chapter III BASIC functions were introduced.

INT(e) -- Returns the nearest integer not greater than the value given in the expression e.

RND(1) -- Returns a different random number each time the function is executed. The number is greater than or equal to Ø and less than 1.

Functions may appear in any of the BASIC statements: LET, PRINT, IF-THEN, and FOR. They may appear freely in expressions and may even be included in the expression for another function. By combining the INT and RND(1) functions, Random Chance Events may be simulated.

Integers can be generated randomly. Examples include:

INT(2 * RND(1))	returns either a Ø or a 1
INT(2 * RND(1) + 1)	returns either a 1 or a 2
INT(1Ø * RND(1) + 1)	returns one of: 1, 2, 3, 4, 5, 6, 7, 8, 9, 1Ø

Several problems and examples were included. These should be studied, entered to the computer and executed before proceeding on to the review quiz.

Four programming statements are needed to generate low resolution graphics. They are as follows:

GR -- Statement which clears the screen to set low resolution graphics. The screen is now a grid of 4Ø points by 4Ø points.

COLOR = x -- Statement which sets the color of the graphics. x can be any numeric expression. Fifteen different colors can be used.

PLOT X,Y -- Statement which allows a spot to be plotted at the given point. The range of X and Y must be from Ø to 39.

HLIN X,Y AT Z -- Statement which plots a horizontal line from point X,Y to point Z,Y.

VLIN X,Z AT Y -- Statement which plots a vertical line from point X,Y to point X,Z.

Review Quiz - Chapter III

1. Write a BASIC program that will print a blue spot at point 2Ø,2Ø using low resolution graphics.

2. The INT function is said to be a Greatest Integer Function. Locate each of the following values on a number line, then locate each of the INT values for these numbers:

   A. 7.6
   B. -2.3
   C. .08
   D. 4.91
   E. -.5

3. On a number line INT(e) is always the first integer to the _____ of the value e. Is there ever an exception to this statement?

4. Write a BASIC program that will draw a low resolution graphics line from point 2,6 to 1Ø,6.

5. What are the possible digits that will be produced by the following statements?

   A. 2Ø LET A = INT(RND(1) * 8)
   B. 2Ø LET B = INT(RND(1) / .5)
   C. 2Ø LET C = INT(12 * RND(1)) - 15
   D. 2Ø LET D = INT (-8 * RND(1) + 5)

6. Write a BASIC program that will print 1Ø random selected low resolution graphics points on the screen. Make each point green.

7. Write a BASIC program that will print 1Ø random color points at point 2Ø,2Ø.

8. Write a BASIC program that will generate random numbers between 3.75 and 8.25, integers between and including 4 and 8.

9. State a general rule on how to generate random integers between any two integers A and B.

1Ø. Write a BASIC program to find the factors of any random integer between 1 and 1ØØ.

11. Write a BASIC program to find the PRIME factors of any random number between 1 and 1ØØ.

12. Given any 3, 4, or 5 digit number, write a program that will print out the digit in the hundredths place.

13. Write a BASIC program to generate random odd integers between 1 and 99.

14. Write a program to print 1Ø random numbers and the INT value of 1ØØ times the random number.

A game of Horse Race can be played by four players. The game consists of a spinner with the numbers 1, 2, 3, and 4 equally dividing the space around the spinner. Each player takes a turn spinning the spinner. If it lands on 1, player one moves his horse 5 spaces by adding 5 to whatever is on his piece of paper. If it lands on 2, player two adds 5 to his piece of paper and so on. The first player to get 25 points wins the game - his horse crosses the finish line first.

This game can be played by writing and running a BASIC program.

15. Write a BASIC statement to simulate "spinning the spinner."

16. Write decision statements to determine which player was selected by the "spinner simulation."

17. Write BASIC statements to keep track of each player's score.

18. Write decision statements to determine if a player has won. These should occur immediately after his score changes.

19. Put all of the pieces together and draw a flowchart for the game of horse race, making additions when needed. Examine your flowchart to make sure it plays the game as described. Write a program from your flowchart and execute it on the computer.

2Ø. What is meant by animation?

Chapter IV

DATA AND STRINGS

4 - 1 Handling Data

Bob runs a lemonade stand in June, July, and August. His stand sells lemonade seven days a week. In the past, a calendar has been used to record each day's sales. However, Bob has to pay sales tax this year and he needs a new way of recording and making monthly reports. Bob has access to a microcomputer at school and has decided to use it to run a report each month on his sales. Let's see how Bob wrote his program.

Old Method of Recording Sales

JUNE

		1 2.5Ø	2 9.ØØ	3 1.25	4 6.5Ø	5 .75
6 11.ØØ	7 2.75	8 4.Ø5	9 .25	1Ø 3.ØØ	11 14.25	12 1.25
13 3.25	14 1.Ø5	15 7.1Ø	16 2.15	17 8.4Ø	18 5.ØØ	19 2.35
2Ø 6.ØØ	21 4.25	22 3.15	23 7.5Ø	24 2.Ø5	25 7.65	26 9.4Ø
27 3.55	28 11.Ø5	29 6.75	3Ø 3.6Ø			

Assigning the sales for each day of the month provides a problem. LET statements and ordinary simple variables would work, but 3Ø of them would be needed for June, 31 of them for July, and 3Ø more for August.

Bob decided to use subscripted variables. Examine the two calendars below showing variable assignments for the month of June.

JUNE

		A	B	C	D	E
F	G	H	I	J	K	L
M	N	O	P	Q	R	S
T	U	V	W	X	Y	Z
A1	B1	C1	C2			

Using LET statements and simple variable names

JUNE

		J(1)	J(2)	J(3)	J(4)	J(5)
J(6)	J(7)	J(8)	J(9)	J(1Ø)	J(11)	J(12)
J(13)	J(14)	J(15)	J(16)	J(17)	J(18)	J(19)
J(2Ø)	J(21)	J(22)	J(23)	J(24)	J(25)	J(26)
J(27)	J(28)	J(29)	J(3Ø)			

Using subscripted variable names

Notice in the second illustration that the variable names each start with a J. The second part of the name (1), for example, corresponds to a day of the month. A calendar may be thought of as a collection of days.

Using this new way of assigning a variable to a day, J(1), J(2), etc., allows the calendar to be thought of as a collection of storage cell names called an ARRAY.

ARRAYS

An ARRAY is a collection of variable names or storage cells. Arrays are given names. In this case an appropriate name would be array J. Arrays are named the same way that single variables are named, for example, M2 array and H array.

Each variable in the collection is named after the array it belongs to and its location within the array. J(1) means the first storage cell in array J. J(2) is the second. This gives us a third and new way of naming variables.

NAMING SINGLE SUBSCRIPTED VARIABLES

A numeric variable is the name of a storage location in the computer's memory. There are two types of numeric variable names.

1. Simple numeric variables are usually named by two letters or less of the alphabet or a single letter followed by a digit.
2. Subscripted variables are named the same way as simple numeric variables (the name of the array it belongs to), followed by a number inside parenthesis (its location within the array).

For an array containing more than 10 locations, a DIMENSION statement identifying the array and the number of locations it contains must appear in the program before the array is used.

DIMENSIONING ARRAYS

general form: LN DIM n(m)

where: n - is the name of the array.

m - sets the maximum number of storage locations to be used in the array. The number used in naming a subscripted variable for this array may not be larger than m.

purpose: The DIM statement warns the computer that an array and its corresponding subscripted variables will be used in the program. It instructs the computer to reserve a collection of storage locations and to give them the name in the statement.

Returning to his program, Bob decided to use the statement:

```
10  DIM J(30)
```

This statement instructs the computer to create a collection of storage cells in the computer's memory named J (for June). This makes it easy to remember what the array contains. The number 30 corresponds to the number of days in the month. Each of the locations will hold a day's sales amount.

GETTING THE DATA INTO THE COMPUTER

If LET statements were used to assign each of the variables: J(1), J(2), . . . J(30) a number, it would also take 30 statements. Bob first tried an INPUT statement and a FOR-NEXT loop as shown below:

```
100  FOR I = 1 TO 30
110  INPUT J(I)
120  NEXT I
```

Statement 110 is executed by the loop 30 times. The variable I in J(I) is replaced by its numeric value each time the loop is executed. Thus, as the

loop is executed statement 11Ø becomes:

```
J(1) WHEN I = 1
J(2) WHEN I = 2
.
.
.
J(3Ø) WHEN I = 3Ø
```

and creates a new variable name. Each time 11Ø INPUT J(I) is executed, a question mark appears at the terminal; Bob types in a day's sales amount, and the number is stored in a different location in array J. Using subscripted variables in this manner in a loop is called INDEXING.

MATRICES

Similar to an ARRAY the MATRIX is a collection of storage cells. Each variable in the collection is named after the row and column it belongs to in the collection. J(4,2) is the storage cell that represents the fourth row and second column in the collection.

NAMING DOUBLE SUBSCRIPTED VARIABLES

A double subscripted variable is named the same as a single subscripted variable with two numbers within the parenthesis separated by a comma. A matrix must be dimensioned if one or both of the storage cells in the columns or rows exceeds 1Ø.

GETTING DATA INTO THE COMPUTER

Data is entered using two nested loops rather than one. The outside loop represents the rows and the inside represents the columns. Consider the example:

```
11Ø  FOR R = 1 TO 2
12Ø  FOR C = 1 TO 2
13Ø  INPUT J(R,C)
14Ø  NEXT C
15Ø  NEXT R
16Ø  END
```

Statement 13Ø is executed four times by the two loops. The variable R and C in J(R,C) each have two numeric values. Thus, as the loop is executed, statement 13Ø becomes:

```
J(1,1) WHEN R = 1 AND C = 1
J(1,2) WHEN R = 1 AND C = 2
J(2,1) WHEN R = 2 AND C = 1
J(2,2) WHEN R = 2 AND C = 2
```

Each time 13Ø INPUT J(R,C) is executed a question mark appears. The first storage cell is J(1,1). If the numeric value 6 is inputted, then J(1,1) = 6. This is continued until all the storage cells are filled.

Three dimensions is available on the APPLE microcomputer but is not discussed in this text. An example would be A(1,2,3).

Bob found that running the program with the INPUT statement, that it not only took a lot of time, but that once in awhile he made typing errors. Each error caused him to start the program over again. He replaced the INPUT statement with a READ statement and placed all of his sales amounts in DATA statements. This not only saved him time, but he could now correct his typing errors before running the program.

READ STATEMENT

general form: LN READ v1, v2, . . . , vi

where: v1, v2, . . . , vi are simple or subscripted numeric variable names.

purpose: The READ statement assigns numbers to variables named in the statement. When executed, it locates a list of numbers created by one or more DATA statements. It then assigns a number (the DATA POINTER is pointing to) to a variable. The DATA POINTER is then advanced to the next number in the list.

DATA STATEMENT

general form: LN DATA n1, n2, . . . , ni

where: n1, n2, . . . , ni are numbers separated by commas.

purpose: The DATA statements supply the numbers to be assigned to variables in READ statements. Before the program is executed, the computer searches for DATA statements, places the numbers (in the order they appear) in a list called a DATA LIST, and sets a pointer (called a DATA POINTER) to the first number in the list. The data is then available to the program as it is executed. Data statements may appear anywhere in a program, since they do their work prior to the program's execution and are ignored while the program is running.

DATA POINTER

The DATA POINTER keeps track of which number is to be assigned to a variable in a DATA LIST. An end of data marker is placed at the end of the list. As values are assigned to variables in READ statements, the DATA POINTER is moved to the next available number in the list. Should a READ statement request a number to be assigned at the time the DATA POINTER is pointing to the END OF DATA marker, the program will be terminated and the error message given. The student must be careful to place enough data numbers in the list to prevent this from happening.

Here are the statements Bob used in his program to assign his daily sales. (Refer to the calendar on page 96.)

```
6Ø  FOR I = 1 TO 3Ø
7Ø  READ J(I)
8Ø  NEXT I
9ØØ  DATA 2.5Ø, 9.ØØ, 1.25, 6.5Ø, .75
91Ø  DATA 11.ØØ, 2.75, 4.Ø5, .25, 3.ØØ, 14.25, 1.5Ø
92Ø  DATA 3.25, 1.Ø5, 7.1Ø, 2.15, 8.4Ø, 5.ØØ, 2.35
93Ø  DATA 6.ØØ, 4.25, 3.15, 7.5Ø, 2.Ø5, 7.65, 9.4Ø
94Ø  DATA 3.55, 11.Ø5, 6.75, 3.6Ø
```

Each time the READ statement is executed a number is assigned to the storage location named by J(I):

```
J(1) = 2.5Ø WHEN I = 1
J(2) = 9.ØØ WHEN I = 2
.
.
.
J(30) = 3.6Ø WHEN I = 3Ø
```

Bob's complete program follows with a few additions. The array T is used to store the amount of tax to be paid each day. The statement at line 8Ø computes the amount of sales tax to be paid ($.Ø4 for each dollar) rounded to the nearest penny. Variable names T1 and T2 are used to total the month's sales and the amount of tax to be paid. Enter Bob's program and execute it on the computer.

```
BOB'S LEMONADE STAND PROGRAM FOR THE MONTH OF JUNE

1Ø  DIM J(3Ø),T(3Ø)
2Ø  PRINT "BOB'S LEMONADE STAND"
3Ø  PRINT "SALES SUMMARY FOR JUNE"
4Ø  PRINT
5Ø  PRINT "DAY", "SALES", "TAX"
6Ø  FOR I = 1 TO 3Ø
7Ø  READ J(1)
8Ø  LET T(1) = INT(1ØØ * (.Ø4 * J(1) + .ØØ5)) / 1ØØ
9Ø  NEXT I
1ØØ  FOR X = 1 TO 3Ø
11Ø  PRINT X,J(X),T(X)
12Ø  LET T1 = T1 + J(X)
13Ø  LET T2 = T2 + T(X)
```

```
14Ø  NEXT X
15Ø  PRINT "_____", "_____", "_______"
16Ø  PRINT "TOTALS", T1, T2
9ØØ  DATA 2.5Ø, 9.ØØ, 1.25, 6.5Ø, .75
91Ø  DATA 11.ØØ, 2.75, 4.Ø5, .25, 3.ØØ, 14.25, 1.5Ø
92Ø  DATA 3.25 1.Ø5, 7.1Ø, 2.15, 8.4Ø, 5.ØØ, 2.35
93Ø  DATA 6.ØØ, 4.25, 3.15, 7.5Ø, 2.Ø5, 7.65, 9.4Ø
94Ø  DATA 3.55, 11.Ø5, 6.75, 3.6Ø
1ØØØ  end
```

sample problem 1:

Write a program to:

a) create an array W containing 15 storage locations;
b) assign the value 1 to each location using a FOR-NEXT loop and a LET statement;
c) terminate the program.

solution:

```
1Ø  DIM W(15)
2Ø  FOR X = 1 TO 15
3Ø  LET W(X) = 1
4Ø  NEXT X
5Ø  END
```

sample problem 2:

Write a program to:

a) create an array T containing 12 storage locations;
b) allow the user to fill the array while the program is executing by using a FOR-NEXT loop and an INPUT statement;
c) terminate the program.

solution:

```
1Ø  DIM T(12)
2Ø  FOR I = 1 TO 12
3Ø  INPUT T(I)
4Ø  NEXT I
5Ø  END
```

sample problem 3:

Write a program to:

a) create an array D containing 1Ø storage locations; (the numbers to be assigned the array's location are: 12.Ø5, 1.27, 13.14, 1.ØØ, 11.ØØ, 6.ØØ, 7.25, 8.14, 9.2Ø, 1Ø.75)
b) assign the numbers to the array D by using a FOR-NEXT loop and READ-DATA statements;
c) print the contents of the array using a second FOR-NEXT loop and a PRINT statement;
d) terminate the program.

solution:

```
10  DIM D(10)
20  FOR J = 1 TO 10
30  READ D(J)
40  NEXT J
50  DATA 12.05, 1.27, 13.14, 1.00, 11.00, 6.00
60  DATA 7.25, 8.14, 9.20, 10.75
70  FOR K = 1 TO 10
80  PRINT D(K)
90  NEXT K
100  END
```

sample problem 4:

Write a program to:

a) create an array C containing 50 storage locations;
b) store the counting numbers 1 through 50 in the array so that C(1) = 1, C(2) = 2, . . . , C(50) = 50 using a FOR-NEXT loop and a LET statement;
c) print the contents of the array C using a FOR-NEXT loop and a PRINT statement;
d) terminate the program.

solution:

```
10  DIM C(50)
20  FOR N = 1
30  LET C(N) = N
40  NEXT N
50  FOR J
60  PRINT C(J)
70  NEXT J
80  END
```

sample problem 5:

Write a program to:

a) create a matrix containing 20 storage locations;
b) store the numeric values 10, 15, 6, 5, 3, 7, 21, 8, 12, 1, 0, 4, 13, 11, 8, 9, 12, 10, 1, 0 in 5 rows and 4 columns using two FOR-NEXT loops with a READ-DATA statement. Use the matrix J(R,C) where R represents the row and C the column;
c) print out the contents of matrix J using a PRINT statement;
d) terminate the program.

solution:

```
100  FOR R = 1 TO 5
110  FOR C = 1 TO 4
120  READ J(R,C)
130  PRINT J(R,C)
140  DATA 10, 15, 6, 5, 3, 7, 21, 8, 12, 1
150  DATA 0, 4, 13, 11, 8, 9, 12, 10, 1, 0
160  NEXT C
170  NEXT R
180  END
```

Exercise 4 - 1

1. Define the following terms:

   A. Array
   B. Subscripted variable
   C. Indexing
   D. Data pointer
   E. End of data marker

2. Give an example of a dimension statement for an array containing 1ØØ items.

3. Write a loop using FOR-NEXT and INPUT for the purpose of assigning 1Ø values to an array M.

4. Write a loop using FOR-NEXT and PRINT to print the entire contents of an array B (1ØØ).

5. From the lemonade stand problem, why did Bob choose to use READ and DATA statements rather than INPUT to assign the sales values to array J?

6. How many storage cells will the statement DIM A(4,6) hold?

7. Given the statement: 1Ø DIM C(2Ø), D(2ØØ), E(15)
   Which of the following statements in the same program would cause an error message to be printed?

   A. 11Ø LET C(2Ø) = 15
   B. 12Ø LET C(5) = C(2Ø) + C(5)
   C. 13Ø LET E(19) = 5 + C(4)
   D. 14Ø LET C(25) = E(16)

8. State a rule about DIM statements and the use of subscripted variables.

9. Write a BASIC program to:

   A. Create a matrix J
   B. Create a data list containing 1Ø numbers: 2, 4, 6, 8, 1Ø, 12, 14, 16, 18, 2Ø.
   C. Assign these variables using 2 rows and 4 columns using a READ statement.
   D. Print the contents of matrix J using a PRINT statement

1Ø. Write a BASIC program to:

   A. Create an array B
   B. Create a data list containing the numbers: 1, 2, 3, 4, 5, 6, 7, 8, 9, 1Ø, 11, 12
   C. Assign these numbers to the array B using a single READ statement
   D. Print the contents of array B using a single PRINT statement

11-12. Study the final program for Bob's lemonade stand carefully. Write a program similar to Bob's for the month of July using the sales in the calendar below:

JULY

				1 8.ØØ	2 4.9Ø	3 1.25
4 11.ØØ	5 5.25	6 5.2Ø	7 6.ØØ	8 1.25	9 1Ø.2Ø	1Ø 2.1Ø
11 2.35	12 .95	13 4.ØØ	14 .4Ø	15 2.4Ø	16 11.5Ø	17 3.25
18 6.ØØ	19 5.4Ø	2Ø 7.ØØ	21 1.5Ø	22 6.55	23 6.75	24 4.9Ø
25 5.35	26 8.ØØ	27 5.3Ø	28 5.7Ø	29 2.75	3Ø 3.15	31 6.15

Remember July has 31 days and June has only 3Ø.

4 - 2 Strings

So far in this manual, only numbers have been assigned to storage locations. Alphabetic and special characters may also be assigned to storage locations as STRINGS.

definition: A STRING is any series of characters. For example,

YES NO MINNEAPOLIS MINNESOTA

are all strings. They are assigned to storage locations by using string variables.

NAMING STRING VARIABLES

A STRING VARIABLE names a storage location where characters are to be stored. They are named in one of the following ways:

1. A single letter of the alphabet followed by a $.
2. A single letter followed immediately by an alphanumeric character. An alphanumeric character is any letter A through Z or any digit Ø through nine. This is followed by a $.
3. Any of the above followed by a $ and a number inside the parenthesis. This is a subscripted string variable. The double subscripted variable can also be used with strings.

examples: A$ CD$ C3$ P$(19) C$(1,2)

String assignments may be added by using LET, INPUT, or READ/DATA statements.

ASSIGNING STRINGS TO VARIABLES

LET

general form: LN LET v$ = "STRING"

where: "STRING" is one or more characters enclosed in quotes.
v$ - is any string variable name.

example:
```
1Ø  LET R$ = "YES"
2Ø  LET C1$ = "NEVADA"
3Ø  LET P$(15) = "STRINGS ARE FUN AND EASY TO LEARN"
```

INPUT

general form: LN INPUT v1$, v2$, . . . , vi$

where: v1$, v2$, . . . , vi$ are string variable names.

example:
```
1Ø  INPUT R$
2Ø  INPUT Z9$, D$
```

RESPONDING TO INPUT

The INPUT statement causes a question mark to be printed and the computer to wait for one or more strings to be entered. The number of strings entered must correspond to the number of variables in the statement. Commas are used to separate two or more entries.

example:
```
1Ø  INPUT A$
?   "PEACHES AND CREAM, SUGAR AND SPICE"
2Ø  INPUT S$,T$(2)
?   UTAH, MAINE
```

In the response to statement 1Ø, quotes were used. Quotes are optional unless commas, spaces, or punctuation characters are to be included in the

string. In the second example the quotes are omitted. In this case, the comma separates the two responses.

The question marks do not tell us what is to be typed - a number, or a string. It is important to print a message describing what is to be typed.

example:

```
5  PRINT "PLEASE TYPE IN THREE NAMES - ONE AFTER EACH
   QUESTION MARK
1Ø  INPUT A$
2Ø  INPUT B$
3Ø  INPUT C$
?   JONES
?   LARSON
?   ANDERSON
```

location	string
A$	JONES
B$	LARSON
C$	ANDERSON

computer's memory

READ/DATA

general form: LN READ v1$, v2$, . . . , vi$

where: v1$, v2$, . . . , vi$ are string variable names.

causes: A string data list (created by data statements containing strings) to be accessed. The data pointer determines which string is assigned to each variable in the READ statement.

general form: LN DATA string1 , string2 , . . .

causes: Prior to program execution, the computer is instructed to create a data list of strings and to position the data pointer to the first time.

example:

```
1Ø  READ C$, V1$, X$(4)
2Ø  DATA THIS , ISN'T SO , BAD
```

NOTE: Care must be taken when writing READ and DATA statements to match the variables in the READ statement with the DATA in the data statements.

example:

```
1Ø  READ A, B$
2Ø  DATA APPLE
3Ø  END
```

would cause an error. The error message would be "SYNTAX ERROR IN 1Ø".

The data must match the order of the variables.

example:

```
1Ø  READ A, B$
2Ø  DATA 1Ø, APPLE
3Ø  END
```

The program above is correctly written. Quote marks are necessary if commas are to be included as part of the string, or if a number is to be used as a string.

example:

```
1Ø  READ 4T$
2Ø  DATA "MARSHALL, MN"
3Ø  END

1Ø  READ C$
2Ø  DATA "136"
3Ø  END
```

VALUES OF PRINTING STRING VARIABLES

general form: LN PRINT v1$, v2$, . . . , vi$

where: v1$, v2$, . . . , vi$ are any string variable names.

discussion: The same rules for the use of the comma and the semi-colon apply to printing strings as they apply to messages and numeric variables. When executed, the computer is instructed to locate the storage location named by the string variable and print the contents of that location.

example:

```
1ØØ  LET A$ = "GEORGE"
2ØØ  LET B$ = "MURPHY"
3ØØ  LET C$ = "WASHINGTON"
4ØØ  PRINT A$, B$, C$
5ØØ  PRINT A$, B$, C$
```

Exercise 4 - 2

1. Which of the following are not legal string variables?

   A$ DC$ B6 Q% T3$ 27$

2. Which of the following are not legal strings?

   2714
   "314"
   HOUSE
   "MARSHALL, MN"
   ST. LOUIS, MO.

3. Write a LET statement to assign your name to a string variable.

4. Write an INPUT statement to request a string to be assigned to a string variable.

5. Give an example of a subscripted string variable.

6. When must quote marks be used in responding to an INPUT statement?

7. When must quote marks be used to enclose a string in a DATA statement?

8. Write a program which:

   A. Asks for your name and address
   B. Allows you to enter your name and address while the program is executing
   C. Prints your name and address
   D. Terminates the program
   E. Show the run of the program

9. Given the READ statement: 1Ø READ A,B$,C
   Write a DATA statement that would satisfy the READ statement.

1Ø. Write a program which will ask you five questions about yourself. After asking the questions, the program should then print a paragraph describing you.

4 - 3 Decisions and Strings

The IF-THEN statement may also contain strings and string variables. Strings may not be compared with numbers.

RELATIONAL OPERATORS

All of the relational operators are available for use in comparing two strings.

Operator	Description
=	Is the string on the left identical, character for character, to the string on the right?
< >	Is the string on the left different from the string on the right?
<	Is the string on the left alphabetically before the string on the right?
>	Is the string on the left alphabetically after the string on the right?
= <	Is the string on the left identical or alphabetically before the string on the right?
= >	Is the string on the left identical or alphabetically after the string on the right?

When comparing strings or string characters, the computer uses a precise order for characters. On the next page is a list of the characters used in BASIC in their alphabetical order.

A LIMITED SET OF CHARACTERS IN THEIR ORDER

NOTE: This table shows letters of the alphabet in their normal alphabetical order. Special characters and spaces are alphabetically before digits. Digits are alphabetically before letters.

+	
-	
*	
/	
(	
)	
$	
=	
	(space)
,	(comma)
.	(decimal point or period)
"	
:	
'	(single quote)
&	
?	
Ø	
1	
2	
.	
.	
.	
9	
A	
B	
.	
.	
.	
Z	

RELATIONAL EXPRESSIONS

A relational string expression consists of:

(string or variable) operator (string or variable)

The relational expression allows the programmer to request a decision to be made by the computer.

example: "A B" = "AB" - Is A B identical to AB? No, because of the space. The computer compares the two strings character by character. If the two strings are not identical, a value of FALSE is assigned to the expression.

IF-THEN

general form: LN IF (string relational expression) THEN m

purpose: The relational expression is evaluated by the computer. If TRUE, control is transferred to the statement with line number m. The IF-THEN statement with strings allows questions to be asked and the responses tested for the correct answer.

example:
```
1Ø  IF "A" < "B" THEN 1ØØ
2Ø  IF "A" > "2" THEN 2ØØ
3Ø  IF "Z" < " " THEN 3ØØ
```

Statement 1Ø would cause line 1ØØ to be executed because A is alphabetically before B.

Statement 2Ø would result in a true condition because A is alphabetically after the digit 2.

Statement 3Ø would result in a false condition because Z is not alphabetically before a space.

Remember: The order is special characters before digits before letters.

STRING ARRAYS

ARRAYS may be created for strings. A string array is named by a letter of the alphabet followed by a $. For example, array 1$. If the array is to contain 1Ø or more items, a DIM statement must appear in the program.

DIM

The DIM statement warns the computer that a string array is to be used on the program. It also sets the maximum sign of the array. The corresponding subscripted variables may not contain a number greater than the array's size.

example:

```
10  DIM A$(100)
```

Would create an array A$. Up to 100 strings can be stored in this array.

STRING APPLICATIONS

problem 1: Assign the states MINNESOTA and MISSISSIPPI to an array S$ and their capitals ST. PAUL and JACKSON to an array C$.

solution 1:

```
10  DIM S$(2), C$(2)
20  LET S$(1) = "MINNESOTA"
30  LET S$(2) = "MISSISSIPPI"
40  LET C$(1) = "ST. PAUL"
50  LET C$(2) = "JACKSON"
60  END
```

solution 2:

```
10  DIM S$(2), C$(2)
20  FOR X = 1 TO 2
30  INPUT S$(X), C$(X)
40  NEXT X
50  END
```

solution 3:

```
5   DIM S$(2), C$(2)
10  DATA MINNESOTA, ST. PAUL, MISSISSIPPI, JACKSON
20  FOR X = 1 TO 2
30  READ S$(X), C$(X)
40  NEXT X
50  END
```

All three solutions result in the assignment to memory:

S$

S$(1)	MINNESOTA
S$(2)	MISSISSIPPI

location *string*

C$

C$(1)	ST. PAUL
C$(2)	JACKSON

location *string*

problem 2: Modify problem 1 to include asking the user the question WHAT IS THE CAPITAL OF followed by the state. Let the user respond. Check the respone telling if it is correct or incorrect.

solution:

```
10  DIM S$(2), C$(2)
20  DATA MINNESOTA, ST. PAUL, MISSISSIPPI, JACKSON
30  FOR X = 1 TO 2
40  READ S$(X), C$(X)
50  NEXT X
55  FOR Y = 1 TO 2
60  PRINT "WHAT IS THE CAPITAL OF "; S$(Y)
70  INPUT R$
80  IF R$ = C$(Y) THEN 110
90  PRINT "WRONG, THE ANSWER IS ": C$(Y)
100  GO TO 120
110  NEXT Y
120  END
```

problem 3: Sorting string data is a useful task the computer can perform. Write a program to assign the letters of the alphabet to an array A$ in a scrambled order. Sort the array in alphabetic order and print the results.

solution:

```
10  DIM A$(26)
20  DATA Z,A,Y,B,X,C,W,D,V,E,U,F,T,G,S,H,R,I,Q,J,P,K,O,N,M,L
30  FOR I = 1 TO 26
40  READ A$(I)
50  NEXT I
60  LET C = 0
70  FOR Y = 1 TO 25
80  IF A$(Y) < A$(Y + 1) THEN 120
90  LET X$ = A$(Y)
100  LET A$(Y) = A$(Y + 1)
110  LET A$(Y + 1) = X$
115  LET C = C + 1
120  NEXT Y
130  IF C > 0 THEN 60
140  FOR Z = 1 TO 26
150  PRINT A$(Z)
160  NEXT Z
170  END
```

In the solution to problem 3, the statement: 80 IF A$(Y) < A$(Y + 1) THEN 120 contains a less than decision. This decision asks the computer to decide: does the string A$(Y) appear before the string A$(Y + 1) in an alphabetic list? If it does, then the two strings do not have to be

switched. If A$(Y) is greater than A$(Y + 1), then the strings are exchanged using the three steps 9Ø - 11Ø.

In the solution the variable C is used as a counter every time a change is made. In the event C = Ø then the list is printed because all the changes have been made.

Exercise 4 - 3

1. Define a string.

2. Which of the following are legal string variable names?

   A. A1
   B. C$
   C. F(1)$
   D. C1$
   E. F$(1)
   F. FG
   G. DF$

3. Write a statement to assign "HOUSE" to the variable A$.

4. Which statements may be used to assign a string to a variable? Give an example of each.

5. When must quote marks be used when responding to: 1Ø INPUT A$

6. It was mentioned in this section that care must be taken when writing READ/DATA statements. The order of the variables and the data must match. Examine the examples below. Which would cause the program to terminate?

   A.
   ```
   1Ø  READ A$, B$, C$
   2Ø  DATA HAT, CAT, RAT
   ```

   B.
   ```
   1Ø  READ A, B, C
   2Ø  DATA 9, 99, 999
   ```

   C.
   ```
   1Ø  READ A$, B, C$, D
   2Ø  DATA HAT 9, RAT, CAT
   ```

   D.
   ```
   1Ø  READ C$, F1, H2, F$
   2Ø  DATA COWS, 21.4, 99.3, 1ØØ, RATS
   ```

   E.
   ```
   1Ø  READ A, B
   2Ø  DATA 999.99
   ```

7. Change each pair of incorrect statements in problem 6 to correct statement pairs. (Assume there are no other data statements in the program.)

8. Sample problem 3 in this section contains a sorting program for alphanumeric data. Use this program to sort the following list of names:

   Adams, John
   Washington, George
   Eisenhower, Dwight
   Jefferson, Thomas
   Lincoln, Abraham
   Roosevelt, Theodore
   Roosevelt, Franklin
   Kennedy, John
   Ford, Gerald
   Nixon, Richard

9-1Ø. Change the string sorting program given in sample problem 3 to a number sorting program. Use 26 numbers of your choice as data and run the program.

Review - Chapter IV

The BASIC statements DIM, READ, and DATA were introduced in Chapter IV.

DIM -- The dimension statement is executed prior to the rest of the program. It alerts the computer that an array and/or a matrix with its corresponding subscripted variables will appear in the program. When executed, the DIM statement instructs the computer to reserve a specified number of storage locations, to give them a common name, and to set a limit to the subscripted variables which may be used to reference the array in the program.

READ -- The READ statement is used to assign numbers and strings to variables during the execution of the program. The READ statement directs the computer to assign values created by DATA statements to variables in the READ statement.

DATA -- The DATA statement builds the DATA list of numbers and/or strings for assignment to variables in READ statements. Since the DATA statement is executed prior to the rest of the program it may appear anywhere in the program. Care must be taken in placing the numbers and strings in the statements. They must match the variables in the corresponding READ statements in type and order.

The following concepts and terms were also introduced:

ARRAYS -- An ARRAY is a group of storage locations sharing a common name. Numeric arrays are named by a single letter of the alphabet. String arrays are named by a single letter followed by A$. Their size is defined by a DIM statement. The individual locations are accessed by using subscripted variables.

MATRICES -- A MATRIX is a group of storage locations sharing a common name. Matrices are similar to arrays but uses a double dimension to signify rows and columns. The individual locations are accessed by using double subscripted variables.

SUBSCRIPTED VARIABLES -- Subscripted variables are names of storage locations in an array. They are named by using the name of the array they reference followed by a number inside parenthesis. The number determines which location within the array is to be referenced. The number portion of the subscripted variable name may not exceed the number used to define that array in the DIM statement.

DOUBLE SUBSCRIPTED VARIABLES -- Double subscripted variables are names of storage locations in a matrix. They are named with two numbers in parenthesis representing the storage location in the matrix.

INDEXING -- Indexing is a simple way of referencing consecutive storage locations in an array. It allows a variable to appear in the subscripted variable name instead of a number. The value assigned to that variable is used to reference the storage location. For example:

```
10  FOR J = 1 TO 10
20  INPUT A(J)
30  NEXT J
```

Demonstrate a quick means of assigning values to the first 10 locations of array A. Each time statement 20 is executed, the value of J determines which array A location is to be assigned a value.

```
J = 1                    A (1) is referenced
J = 2                    A (2) is referenced
```

This provides a fast and efficient way of assigning values to arrays and printing the contents of arrays.

STRINGS -- The assignment and use of "non-numeric data" can occur in BASIC programs by use of strings. Strings are groups of one or more non-numeric characters. Strings may be assigned to storage locations, printed, and compared.

STRING ASSIGNMENT -- Strings are assigned to string variables by the LET, INPUT, and READ/DATA statements. String variables are named in one of the following ways:

1. A letter of the alphabet followed by a $.

2. A single letter of the alphabet followed by an alphanumeric character. An alphanumeric character is any letter A through Z and any digit 0 through 9. This is followed by a $.

3. A subscripted variable followed by a $.

4. A double subscripted variable followed by a $.

String variables may appear in assignment statements. For example:

```
10  LET A$ = "STRING"
```

Quotes must enclose strings in LET statements:

```
20  INPUT A$
```

Strings may be entered without quotes in response to INPUT unless punctuation marks are used in the string.

```
30  READ A$
```

Review Quiz - Chapter IV

1. Match the term on the left with its corresponding example on the right:

simple numeric variable	I(R,C)
simple string variable	L(K)
subscripted numeric variable	B$
subscripted string variable	B$(5)
array	M(22)
matrix	C
double subscripted numeric variable	G(2,1)

2. Write a statement to define an array A containing 40 storage locations.

3. Write a statement to print the content of the 22nd location of an array M$.

4. What statement in a program would you look for to see how large a number could be used for a subscripted variable name?

5. Write a READ and DATA statement to make the following assignments:

   15 assigned to C1
   CAT assigned to C$
   25 assigned to R(1)
   DOG assigned to B$

6. Where may you expect to see DATA statements in a program?

7. Use the technique of indexing to print the contents of the first 20 items in an array H.

8. How are string variables named?

9. Write a statement to make the decision is A$ equal to YES.

10. Write a complete program to:

    A. Define an array C$. The array should have as many locations as there are students in your English class.
    B. Use READ, DATA, and FOR-NEXT statements to assign the name of each student in your English class to location in array C$.
    C. Sort the names of the students in your class in alphabetical order.
    D. Print the contents of the sorted array C$ at the microcomputer.

Chapter V

SPECIAL FEATURES

5 - 1 Screen Formatting

The output screen for the APPLE microcomputer is 24 lines with 4Ø characters on each line. The output can be printed in different locations on the screen depending on the types of commands used.

To move vertically up or down on any of the 24 lines use the VTAB command.

general form: LN VTAB e

where: e may be any number 1 through 24.

example: 1ØØ VTAB1Ø

purpose: To print text on any of the 24 available lines of output. In the example text would be printed on line 1Ø.

sample:
```
1ØØ  VTAB1Ø
11Ø  PRINT "THIS IS A SAMPLE"
12Ø  END
```

result: The statement THIS IS A SAMPLE would be printed on the 1Øth line starting in the first column. If a number outside the range of 1-24 is used, the message ILLEGAL QUANTITY ERROR will be printed.

To move horizontally left or right to any of the 4Ø columns use the HTAB command.

general form: LN HTAB e

where: e may be any number 1 through 255.

example: 1ØØ HTAB2Ø

purpose: To print text starting at a given column. In the example text would be printed starting in column 2Ø.

sample:
```
1ØØ  HTAB2Ø
11Ø  PRINT "SAMPLE SET"
12Ø  END
```

result: The statement SAMPLE SET would be printed starting in the 2Øth column.

Any positive position greater than 4Ø will result in print starting on subsequent lines depending on the position up to and including column 255. So columns 41 through 8Ø would appear on the next line and so on. A negative number or a number greater than 255 would result in the message ILLEGAL QUANTITY ERROR.

sample:
```
1ØØ  VTAB1Ø
11Ø  HTAB5
12Ø  PRINT "CENTERING THE PAGE"
13Ø  END
```

result: The text CENTERING THE PAGE will be printed on the tenth line starting in column 5.

To move to the right to any of the 4Ø columns use the TAB command. The TAB command must be used in the PRINT statement. The argument must also be in parenthesis. TAB moves to the column on the line starting with the left column. TAB moves printing only to the right whereas HTAB moves printing left or right. TAB(Ø) will move to position 256.

general form: LN PRINT TAB(e)

where: e may be any number Ø through 255.

example:
```
1ØØ  PRINT TAB(1Ø) "SAMPLE"
```

purpose: To print text starting at a given column. In the above example SAMPLE will be printed in columns 1Ø through 15.

sample:

```
100  PRINT TAB(20) "SAMPLE"
110  PRINT TAB(60) "PRODUCT"
120  END
```

result: The statement SAMPLE will be printed on the line starting in column 20. A line will be skipped and PRODUCT will be printed on the next line starting in column 20.

Any argument less than the present column will force printing to start at the present column. A negative argument or a number greater than 255 would result in the message ILLEGAL QUANTITY ERROR.

To skip spaces before printing use the SPC command.

general form: LN PRINT SPC(e)

where: e may be any number 0 through 255.

example:

```
100  PRINT SPC(10) "SAMPLE"
```

purpose: To skip positions before the text is printed. In the example ten positions will be skipped before SAMPLE is printed.

sample:

```
100  PRINT SPC(5) "SAMPLE"
120  END
```

result: The statement SAMPLE would be printed starting in the 9th position as SPC starts in the zero position.

sample:

```
100  PRINT TAB(5) "SAMPLE"; SPC(10) "PRODUCT"
110  END
```

result: The statement SAMPLE would be printed in the 5th position, 10 columns will be skipped and PRODUCT will be printed starting in the 21st position.

The argument can be any position number 0 through 255. Any other number will result in the message ILLEGAL QUANTITY ERROR.

Exercise 5 - 1

1. Describe the use of the VTAB command.

2. Describe the use of the HTAB command.

3. Describe the use of the TAB command.

4. Describe the use of the SPC command.

5. Write a BASIC program that will print the message CENTER OF SCREEN on line 12 and starting in column 12.

6. What column will the letter P be printing using the following BASIC statement?

```
1ØØ  PRINT TAB(8) "SAMPLE"; TAB(2Ø) "PICTURE"
```

7. Write a BASIC statement using the SPC command that will start printing in column 25.

8. Write a BASIC program that will print the following:

```
                    column 1Ø
line 11             SAMPLE MESSAGE
     12                SAMPLE MESSAGE
     13                   SAMPLE MESSAGE
```

9. Describe where the message THIS IS MY PROGRAM will be printed on the screen (line and column):

```
1ØØ  VTAB5
11Ø  HTAB5Ø
12Ø  PRINT "THIS IS MY PROGRAM"
13Ø  END
```

1Ø. Which of the following are legal statements?

A. VTAB26
B. HTAB26
C. PRINT TAB4 "TIME"
D. PRINT SPC(Ø) "TIME"
E. HTAB25Ø
F. VTAB(3)

5 - 2 High Resolution Graphics

general form: HGR with no argument

purpose: This statement clears the screen to black and sets high resolution graphics. High resolution graphics turns the screen into a grid of 28Ø horizontal points by 16Ø vertical points. Again four lines are set for normal printing.

Before any of the points can be plotted on the screen, the color to be used must be set. The colors that can be used and their corresponding equivalent numbers are:

Ø = black
1 = green
2 = violet
3 = white

4 = black
5 = orange
6 = blue
7 = white

general form: HCOLOR = X

where: X is one of the above numbers. X can be any numeric expression.

purpose: When executed the color is set so plotting of points can take place.

The process of plotting is described below:

general form: HPLOT X,Y

where: X is the X-coordinate and Y the Y-coordinate. The range for X is from Ø to 279. The range for Y is from Ø to 159.

Point Ø,Ø is in the upper left-hand corner and 279,159 is in the lower right-hand corner of the screen.

example:

```
10  HGR
20  HCOLOR = 2
30  HPLOT 140,80
40  END
```

Enter and run the above program. The program should plot a violet point at point 140,80 which is about at the center of the screen.

example:

```
10  HGR
20  HCOLOR = 2
30  HPLOT 20,20 TO 260,20
40  HPLOT 260,20 TO 260,140
50  HPLOT 260,140 TO 20,140
60  HPLOT 20,140 TO 20,20
70  END
```

Enter and run the above program. The program should plot a violet border to form a rectangle. Lines 30 through 60 could be suppressed into one statement:

```
30  HPLOT 20,20 TO 260,20 TO 260,140 TO 20,140 TO 20,20
```

The above step saves a lot of programming steps.

In order to clear the screen and move the cursor (blinking square) to the top left of the screen, use the following:

general form: HOME with no argument

purpose: When executed the cursor is moved to the top left of the screen.

example:

```
10  HGR
20  HCOLOR = 2
30  HPLOT 160,0 TO 160,159
40  GET A$
```

```
5Ø  TEXT
6Ø  HOME
7Ø  END
```

Enter and run the above program. Line 4Ø requests an input. Lines 5Ø and 6Ø will clear the screen of graphics and the cursor will be flashing at the upper left of the screen.

Exercise 5 ← 2

---

1. Describe the screen layout for the command HGR.

2. Describe the use of the HOME command.

3. Write a program that will plot two orange squares in the middle part of the screen. The squares should have a middle common side. Make each square 4Ø HGR units on each side.

4. Write a BASIC statement that will randomly select one of the colors used in high resolution graphics.

5. Add two steps to the following program that will clear the graphics from the screen and move the blinking cursor to the top left corner of the screen:

```
11Ø  HGR
12Ø  HCOLOR = 3
13Ø  HPLOT 15,4Ø AT 5Ø
14Ø  END
```

Review - Chapter V

In this chapter several of the special features of the APPLE microcomputer were discussed. The first part included several ways to print information on the output screen.

VTABe -- enables text to be printed on any of the twenty-four lines. e can be any value 1 through 24.

HTABe -- enables text to be printed on any of the forty columns. e can be any value 1 through 255.

TAB(e) -- the TAB command must be used with a PRINT statement. TAB moves printing only to the columns in the right position. e can be any value Ø through 255. e is counted from the left column.

SPC(e) -- the SPC command must be used with a PRINT statement. SPC skips columns starting with the present position.

sample:

```
1ØØ  VTAB6
11Ø  HTAB2Ø
12Ø  PRINT "SAMPLE";SPC(4);"TEST"
13Ø  END
```

result: The message SAMPLE will be printed on the sixth line starting in the 2Øth column. Four columns will be skipped and the message TEST will be printed starting in the 3Øth column.

sample:

```
1ØØ  PRINT TAB(5);"THIS IS MY"
1Ø5  PRINT TAB(5);"STORY.     "
```

```
125  PRINT TAB(1Ø);"PRESS ANY KEY TO CONTINUE"
13Ø  GET A$
14Ø  END
```

result: The TEXT using left-hand margins will be held on the screen until any key is pressed.

Three programming statements needed to generate high resolution graphics are:

HGR -- statement which clears the screen to set high resolution graphics. The screen is now a grid of 28Ø horizontal points by 16Ø vertical points.

HCOLOR = X -- statement which sets the color of the graphics. X can be any numeric expression. Seven different colors can be used.

HPLOT X,Y -- statement which allows a spot to be plotted at X,Y. The range for X is Ø to 279 and the range for Y is Ø to 159.

Besides the TEXT command that is used to clear the screen back to TEXT mode, the HOME command is used to clear the screen of text and the blinking cursor will appear in the left-hand corner of the screen.

TEXT -- clears the graphics screen.

HOME -- moves the cursor (blinking square) to the upper left corner of the screen.

Review Quiz - Chapter V

1. Describe the difference between HTAB1Ø and TAB(1Ø).

2. Describe the difference between TAB(2Ø) and SPC(2Ø).

3. Describe the output of the following program:

```
1ØØ  VTAB6
11Ø  HTAB5Ø
12Ø  PRINT "SAMPLE"
13Ø  END
```

4. How many colors can be used with high resolution graphics?

5. Describe the use of the TEXT command.

6. Describe the use of the HOME command.

7. Describe the grid used in high resolution graphics.

8. Describe the output of the following program:

```
11Ø  HGR
12Ø  HCOLOR = 5
13Ø  HPLOT 8Ø,4Ø TO 12Ø,4Ø
14Ø  HPLOT 12Ø,4Ø TO 12Ø,8Ø
15Ø  HPLOT 12Ø,8Ø TO 8Ø,8Ø
16Ø  HPLOT 8Ø,8Ø TO 8Ø,4Ø
```

MECC INSTRUCTIONAL SERVICES ACTIVITIES

PURPOSE: The primary purpose of the Minnesota Educational Computing Consortium (MECC) is to assist users and educational member systems in the coordination and utilization of computing resources through cooperative planning and decision making, and to provide current computing methods and materials.

SERVICES: All MECC activities in the area of instructional computing are the responsibility of the Director of Instructional Services (telephone **612/376-1105**). Questions related to MECC policy, procedures, or regulations should be directed to this office. The MECC Instructional Services Division is organized as follows:

Instructional Systems Development - This group is responsible for the production coordination, refinement, and distribution of MECC instructional computing courseware products, computer programs and their related user support materials. Questions on operations within this area should be directed to the Manager, Instructional Systems Development (telephone **612/376-1103**).

Technical Services - This group is responsible for operation and operating systems maintenance of the MECC Timeshare System (MTS), a 400+ port, all purpose, multiple language computer, which serves all of Minnesota public higher education institutions and 340 school districts. Microcomputer programming and system utilities are handled by this group. Questions on operations within this area should be directed to the Manager, Technical Services (telephone **612/376-1141**).

User Services - This group is responsible for timeshare and microcomputer user communications and training, the distribution of computing equipment, and the establishment and maintenance of the MTS telecommunications network. A staff of instructional computing coordinators are located throughout Minnesota for the purpose of promoting and facilitating computer usage. All questions on operations in this area should be directed to the Manager, User Services (telephone **612/376-1101**).

GENERAL INFORMATION: The above information is to assist individuals who wish to contact the MECC office with specific questions. All written requests for information should be addressed to the appropriate office at MECC, 2520 Broadway Drive, St. Paul, MN 55113. THE FOLLOWING TWO ITEMS ADDRESS MANY ROUTINE QUESTIONS:

MECC PUBLICATIONS AND PROGRAMS PRICE LIST
No Cost. Distributed upon request. Suggest obtaining on a quarterly basis. Contact: MECC Publication's Office (telephone **612/376-1118**).

MECC USERS Newsletters
No Cost. Distributed regularly during school year to individuals on mailing list. Contact: User Services Office (telephone **612/376-1117**).

All requests for visitations to MECC must be scheduled in advance by calling **612/376-1130.**

COMMENT SHEET

Manual Title: _____

_____

Publication No. _____ Revision Date _____

FROM: Name: _____

School System: _____

Address: _____

_____

_____ ZIP _____

COMMENTS:

Your evaluation of this manual will be welcomed by MECC. Any errors, if called to our attention, can be corrected in future printings. Suggested additions or deletions can be put down here. For all comments please include the page number and paragraph reference.

To Mail: Fold on dotted lines,
staple and stamp.

STAPLE

STAPLE

FOLD

FOLD

First Class
Postage
Needed

Minnesota Educational Computing Consortium
MECC Publications
2520 Broadway Drive
St. Paul, MN 55113

FOLD

FOLD