System 80

PROGRAMMING FOR BEGINNERS

TABLE OF CONTENTS

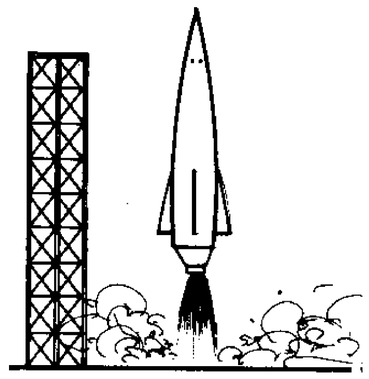
PREFACE

This manual is designed as an introduction to computer programming for beginners knowing little or nothing about computers. All of the main computer programming concepts and techniques will be discussed in detail, and further explained with plenty of real life examples.

No matter what you now think a computer is like , and what it does, by the time you finish reading this manual, you will find out a computer is nothing but a man-made machine – very powerful of course. And programming is nothing more than implementing a sequence of logical steps into instructions that the computer can understand. With the help of our BASIC Manual, you can easily make the System 80 work for you as you wish.

INTRODUCTION

What is computer programming? Programming is the method used to analyze a problem, translate the problem into computer language, and let the computer to solve the problem.



It is obvious that the problem solving ability is limited by the followings:

1. Your ability to analyze a problem.
2. The capability of the programming language you use.
3. The power and facilities of the computer itself.

1. THE PROBLEM ITSELF

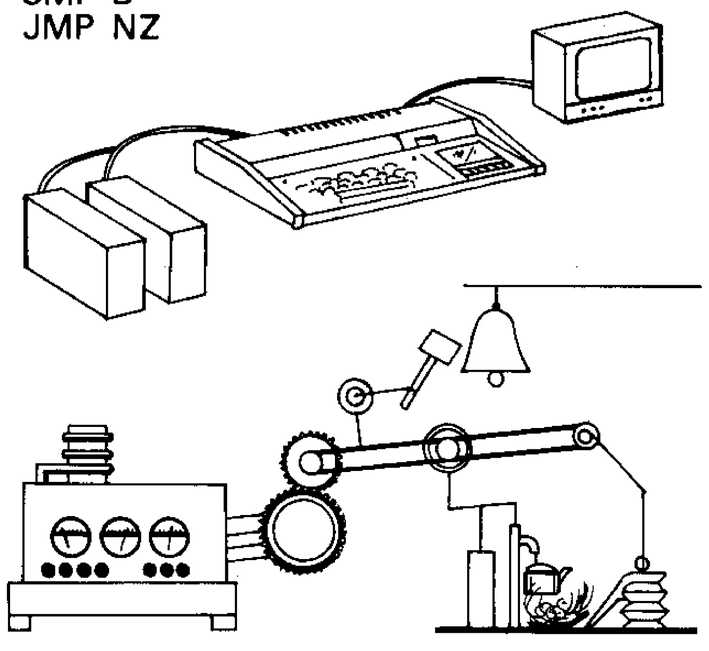
To be more precise, if you know nothing about space missiles, it is very unlikely that you can implement a space missile simulation program that is meaningful to an expert. Unless an expert is patient enough to explain all the factors and variables involved in space missiles to you with his spare time!

```
IF A = B
   THEN A = A-1
```

2. THE PROGRAMMING LANGUAGE ITSELF

The functions and features of a programming language reflect its capability. Some are devoted to business usage, some are for scientific computation, some are for general purpose, of course. For example, the Extended Basic in our System 80 is one of the most popular general purpose computer languages available.

```
LD   A, 3CH
CMP B
JMP NZ
```



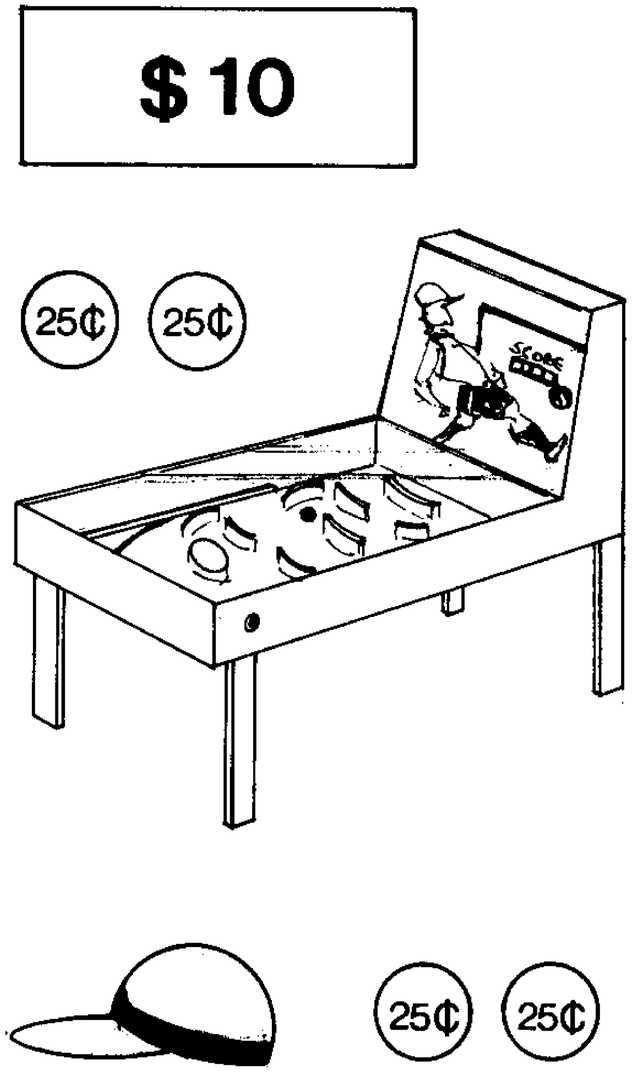
3. THE COMPUTER ITSELF

Finally, the speed and architecture of the computer itself are very important. If your computer is twice as fast as others, that means you can finish the job in half the time other people have spent. Futhermore, if the memory size of your computer is relatively large, you can write longer, more powerful programs.

Suppose we want to solve the following problem:

Example 1

1. David has $10 in his pocket.
2. He took a bus to the mall, the fare is 50¢.
3. He bought an ice cream cone for 89¢.
4. He played 5 games of pin ball, at 25¢ per game.
5. He bought a football fan cap for $3.99.
6. He took a bus back home; the fare is 50¢.



Problem 1: How much money does David has when he arrives home?

If the above events are implemented into our Extended Basic language, we may create the following program:

Program 1

```
10 D = 10              :REM  1. DAVID HAS $10
20 D = D - 0.5         :REM  2.  MINUS BUS FARE 50 CENTS
30 D = D - 0.89        :REM  3.  MINUS ICE CREAM COST 89 CENTS
40 D = D - 0.25 * 5    :REM  4.  MINUS 5 GAMES OF PIN BALL, @ 25 CENTS
50 D = D - 3.99        :REM  5.  MINUS CAP COST $3.99
60 D = D - 0.5         :REM  6.  MINUS BUS FARE 50 CENTS
70 PRINT "DAVID HAS $"; D    :REM  PRINT OUT THE AMOUNT DAVID HAS
80 END                 :REM  END OF PROGRAM
```

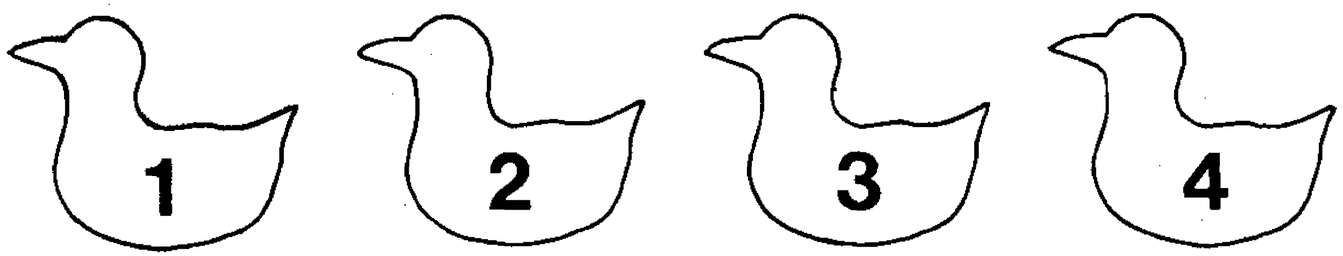
Everytime you RUN this program, the result will be `DAVID HAS $ 2.87`

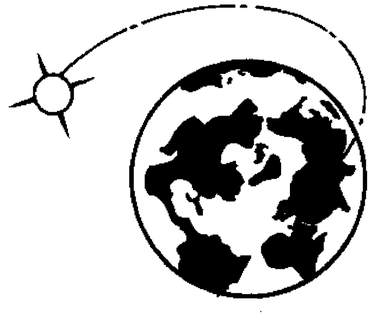
Actually, Program 1 can be reduced to the program below, however, it is much harder to follow.

Program 2

```
10 PRINT "DAVID HAS $";10-0.5-0.89-0.25*5-3.99-0.5
20 END
```

The function of a computer is not limited to do the problem described in program 1. These calculations can be handled by a simple calculator or even by a small child. In the next few chapters, we will introduce more programming concepts, together with some powerful programs, which can solve some really interesting and far more difficult problems.





ACTIVE COMMANDS AND TEXT EDITING

The main purpose of Active Commands is to order the computer to do some work internally and externally.

Whenever a READY message appears on the screen (see User's Manual for system start up and operation), you are in the Active Command level and the computer is waiting for your command. It will wait forever until you enter some instructions throught the keyboard!

Some of the valid Active Commands are:

AUTO	turn on automatic line numbering
DELETE	delete program line
EDIT	Edit program line
LIST	list all program lines
RUN	Execute the program.

Now let us try to enter the following underlined statements into the computer through the keyboard.

Program 3

```
READY
>AUTO    NEW LINE
10 PRINT "MY FIRST PROGRAM"    NEW LINE
20 FOR A = 1 TO 10    NEW LINE
30 PRINT A, A * A    NEW LINE
40 NEXT A    NEW LINE
50 END    NEW LINE
60 -  BREAK
>
```

At this point, check for typing errors. For example, if you made a mistake in line 30, then you must retype the underlined portion:

```
READY
30  PRINT A, A * A    NEW LINE
```

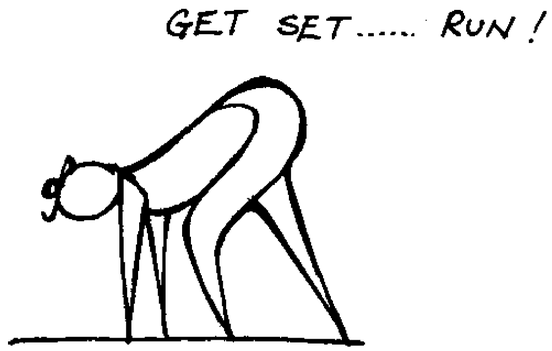
Then line 30 will have the new edited text.
In the next chapter, we will discuss more efficient Text Editing techniques.
If everything is set, we can run the program and see what happens.

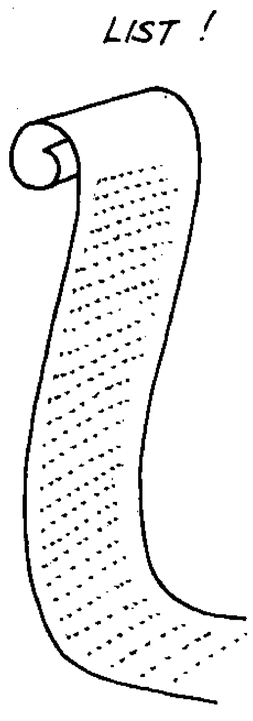
```
READY
>RUN    NEW LINE
MY FIRST PROGRAM
1                 1
2                 4
3                 9
4                 16
5                 25
6                 36
7                 49
8                 64
9                 81
10                100
READY
>
```

Within the entire operation above, only two commands are used, AUTO and RUN. The other parts are just program text and output.

Now type in LIST and press the NEW LINE key.

```
READY
>LIST    NEW LINE
10 PRINT "MY FIRST PROGRAM"
20 FOR A = 1 TO 10
30 PRINT A, A * A
40 NEXT A
50 END
READY
>
```





The entire program will be displayed on the screen by using the LIST command.
Now do you have any idea about what commands are used for in computers?
There are many such Active Commands available for the System 80. Please refer to the BASIC MANUAL for more detail.

TEXT EDITING

We always have the inclination to make mistakes. It may not be too bad, as long as we can detect, and correct our mistakes fast.

The function of Text Editing in the System 80 is to allow the users to input, correct and update their programs in the most efficient manner possible.

Some of the Editing features are:

L	(list)
I	(insert)
C	(change)
D	(delete)

For the complete set of Editing features, please refer to the BASIC Manual, Chapter 2.

Now let us go back to Program 3.

```
READY
>LIST    NEW LINE
10 PRINT "MY FIRST PROGRAM"
20 FOR A = 1 TO 10
30 PRINT A, A * A
40 NEXT A
50 END
READY
>
```

If, for example, you want to change the text of line 30 : change A * A to A * 2.
All you have to do is the following:

```
>READY
>EDIT  30    NEW LINE
30 _
```

Now hit the L key, the entire line will be displayed on the screen.

```
30 PRINT A, A * A
30 _
```

CHANGE A*A TO A*2

Typing the [SPACE BAR] once, the cursor will move a space to the right. Now, let us move the cursor to the 12th position.

```
30 PRINT A, A * _
```

Then you hit the [C] key followed by 2.

```
30 PRINT A  A * [C] 2
```

Once you hit the [NEW LINE] key, the computer will record all the changes you made, and return to the Active Command level.

```
30 PRINT A, A * 2    [NEW LINE]
>_
```

This time if you run the program, the output should be the value of A times two, instead of the squares of A. (A has the value of 1 to 10)

Let us list the program once more.

```
>LIST
10 PRINT "MY FIRST PROGRAM"
20 FOR A = 1 TO 10
30 PRINT A, A * 2
40 NEXT A
50 END
READY
>
```

If we want to change line 10 a little.

```
>EDIT 10     NEW LINE
10 L
```

Line 10 will be displayed on the screen:

```
10 PRINT "MY FIRST PROGRAM"
10 _
```

Now type in 7 followed by hitting the SPACE BAR

```
10 PRINT "MY FIRST PROGRAM"
10 7    SPACE BAR
```

The display will become

```
10 PRINT "MY FIRST PROGRAM"
10 PRINT "7_
```

Hit the I key followed by **.

```
10 PRINT "MY FIRST PROGRAM"
10 PRINT " I **
```

If you hit the NEW LINE key, the display should become

```
10 PRINT "MY FIRST PROGRAM"
10 PRINT "** MY FIRST PROGRAM"
```

Now, you should have some ideas about text editing. Suppose you don't want other people to know this is your first program. So you want to change the program again!

```
>EDIT 10     NEW LINE
10 L
```

The display should be

```
10 PRINT "**'MY FIRST PROGRAM"
10 _
```

Move the cursor to the 13th position.

```
10 PRINT "** MY FIRST PROGRAM"
10 PRINT "** MY _
```

Type in 6 followed by D

```
10 PRINT "** MY FIRST PROGRAM"
10 PRINT "** MY! FIRST!
```

"**MY ~~FIRST~~ PROGRAM"

Everything between the two! marks will be deleted.
Now you hit the NEW LINE key.

```
10 PRINT "** MY FIRST PROGRAM"
10 PRINT "** MY! FIRST! PROGRAM"
>_
```

Now you are in the Active Command level again, so you can list line 10.

```
>LIST 10    NEW LINE
10 PRINT "** MY PROGRAM"
```

That means, when you run the program, the heading will be "** MY PROGRAM", instrad of "** MY FIRST PROGRAM".

The next step is to read the Basic Manual, and get familiar with other Active Commands and Editing features.

BASIC PROGRAMMING

In this chapter we will discuss how to analyze a problem and how to implement it into Extended Basic – our future language.

Before we go further, please study example 1 in the Introduction.

Now let us try to analyze the procedure required to compute the area and circumference of a circle with radius R.

The area of a circle can be found by using the formula $A = \pi R^2$. And the circumference can be determined by the relationship, $C = 2\pi R$.

The diagram shown is called a flow-chart. The direction of flow is indicated by the arrows. The START and END indicates the starting and finishing point of the program, respectively. Other operations include INPUT, OUTPUT, and ASSIGNMENT, etc.

The flow-chart shown will ask for an input, R. With R as the radius, compute the area (πR^2), and the circumference ($2\pi R$). Then the print outs will be the values of the radius, the area, and the circumference.

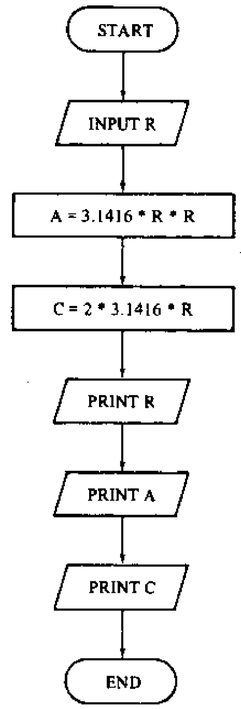
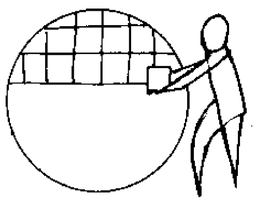
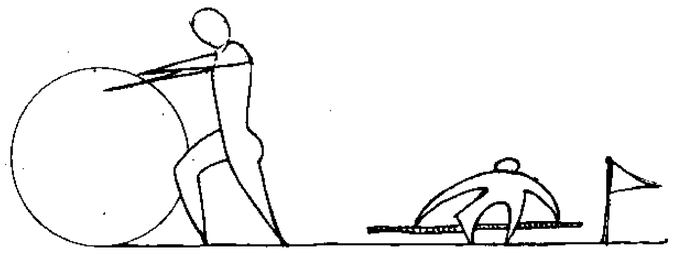


FIG. 3.1

Now we can implement the analyzed problem into Extended Basic, and enter the program into the computer.

```
10 INPUT "ENTER THE RADIUS"; R
20 A = 3.1416 * R * R
30 C = 2 * 3.1416 * R
40 PRINT "THE RADIUS:"; R
50 PRINT "THE AREA IS:"; A
60 PRINT "THE CIRCUMFERENCE IS:"; C
70 END
READY
>RUN
```





```
ENTER THE RADIUS ? 5
THE RADIUS: 5
THE AREA IS: 78.54
THE CIRCUMFERENCE IS: 31.416
READY
>
```

(for example, you type in 5 followed by hitting the NEW LINE key.)

If we want to try another number, we must RUN the program again. Very inconvenient, right?

Let us change the flow chart a little in order to solve this problem.

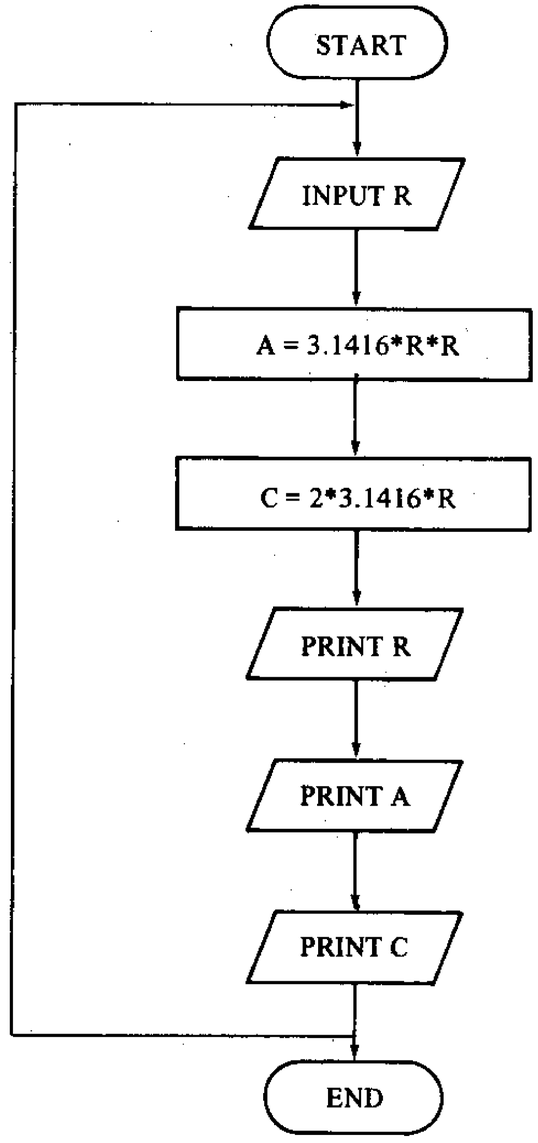


FIG 3.2

As the flow chart shows, everytime the computer finished printing the value of C, it goes back to the beginning of the chart, asks for the value of R, and loops round and round forever.

Again, we can translate the flow chart's logic into Extended Basic.

```
10 INPUT "ENTER THE RADIUS"; R
20 A = 3.1416 * R * R
30 C = 2 * 3.1416 * R
40 PRINT "THE RADIUS:"; R
50 PRINT "THE AREA IS:"; A
60 PRINT "THE CIRCUMFERENCE IS:"; C
70 GOTO 10
80 END
READY
>RUN
ENTER THE RADIUS ? 7
THE RADIUS: 7
THE AREA IS: 153.938
THE CIRCUMFERENCE IS: 43.9824
ENTER THE RADIUS ? 10
THE RADIUS: 10
THE AREA IS: 314.16
THE CIRCUMFERENCE IS: 62.832
ENTER THE RADIUS ?_
•
•
```

The computer repeats in these sequence, until you hit the BREAK key which interrupts the further execution of the program.

(It is not a good programming practice to use the BREAK key, because it forces the unnatural termination of a programs.)

Let us try something more logical. Again, we modify the flow chart a little.

Don't panic ! Try to compare FIG. 3.3 with FIG. 3.1 and 3.2. There is not much changed, right?

The revised program in Extended Basic should be:

```
10 INPUT "ENTER THE RADIUS"; R
15 IF R <= 0 THEN END
20 A = 3.1416 * R * R
30 C = 2 * 3.1416 * R
40 PRINT "THE RADIUS."; R
50 PRINT "THE AREA IS."; A
60 PRINT "THE CIRCUMFERENCE IS."; C
70 GOTO 10
```

This program accepts the value of R, if R is less than or equal to (" <= ") zero, then stops the program. Otherwise, it computes the area and circumference of the circle, prints out the values, goes back to line 10, and ask for another value of R.

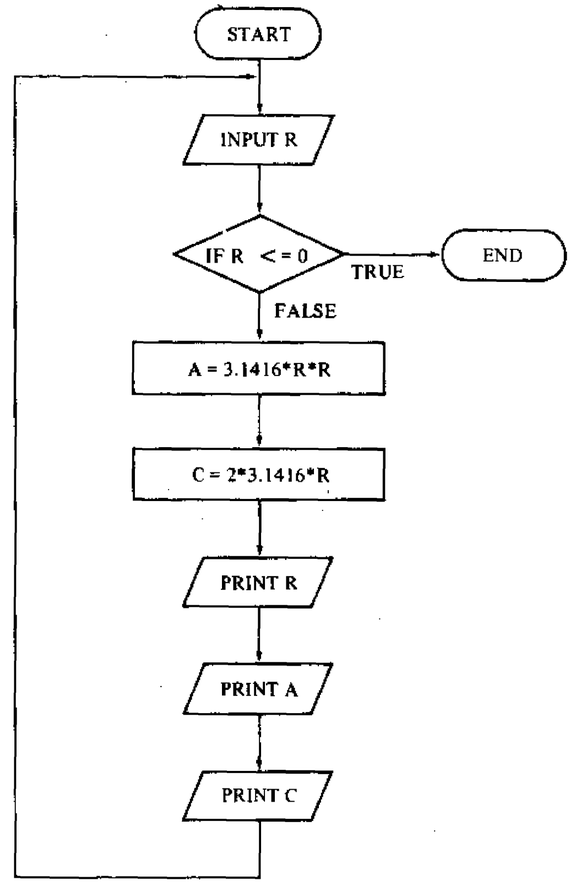
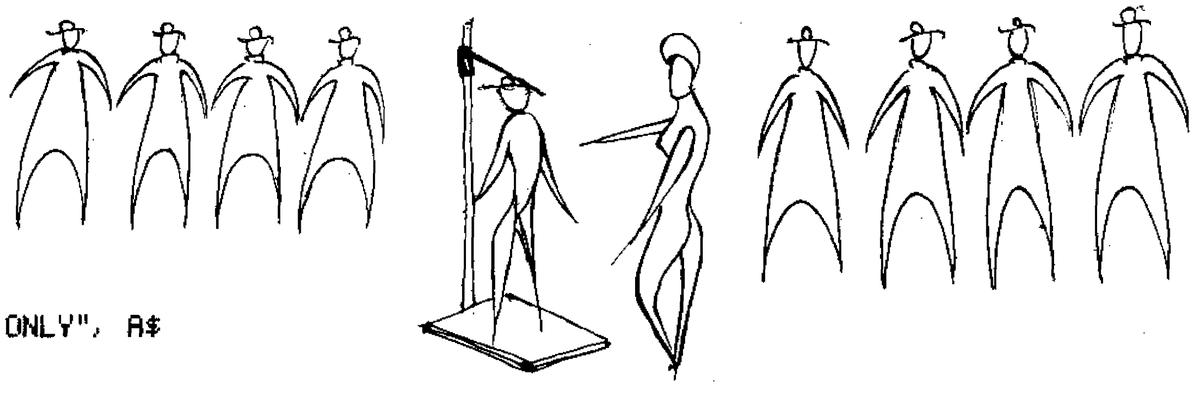


FIG. 3.3

Actually, there are some more "human oriented" methods to terminate a program, other than inputting a code number. Let us go through the program below.

```
10 INPUT "ENTER THE RADIUS"; R
20 A = 3.1416 * R * R
30 C = 2 * 3.1416 * R
40 PRINT "THE RADIUS:"; R
50 PRINT "THE AREA IS:"; A
60 PRINT "THE CIRCUMFERENCE IS:"; C
70 PRINT
80 PRINT "DO YOU WANT TO CONTINUE ?"
90 INPUT "PLEASE ENTER 'YES' OR 'NO' ONLY", A$
100 IF A$ = "YES" GOTO 10
110 IF A$ = "NO" THEN END
120 GOTO 80
```



Just like before, the program asks for the radius, then computes the area and circumference of the circle. It then asks if you want to continue or not. Enter "YES" or "NO". Don't be scared by the $ sign in A$ in line 90. All it does is to inform the computer that the input should be handled as a string (a group of characters or symbols), instead of digits (figures). For detailed explanation about string variables, please refer to the BASIC Manual.

Now, try to understand the flow-chart shown :

Try to run the program:

```
READY
>RUN
ENTER THE RADIUS ? 8
THE RADIUS: 8
THE AREA IS: 201.062
THE CIRCUMFERENCE IS: 50.2656
DO YOU WANT TO CONTINUE ?
PLEASE ENTER 'YES' OR 'NO' ONLY ?YES
ENTER THE RADIUS ? 12
THE RADIUS: 12
THE AREA IS: 452.39
THE CIRCUMFERENCE IS: 75.3984
DO YOU WANT TO CONTINUE ?
PLEASE ENTER 'YES' OR 'NO' ONLY ?N
DO YOU WANT TO CONTINUE ?
PLEASE ENTER 'YES' OR 'NO' ONLY ?NO
READY
>
```

Now, do you have a better idea how one single statement can change the capability of a program? There are many powerful statements in our Basic Manual. Be sure you read them.

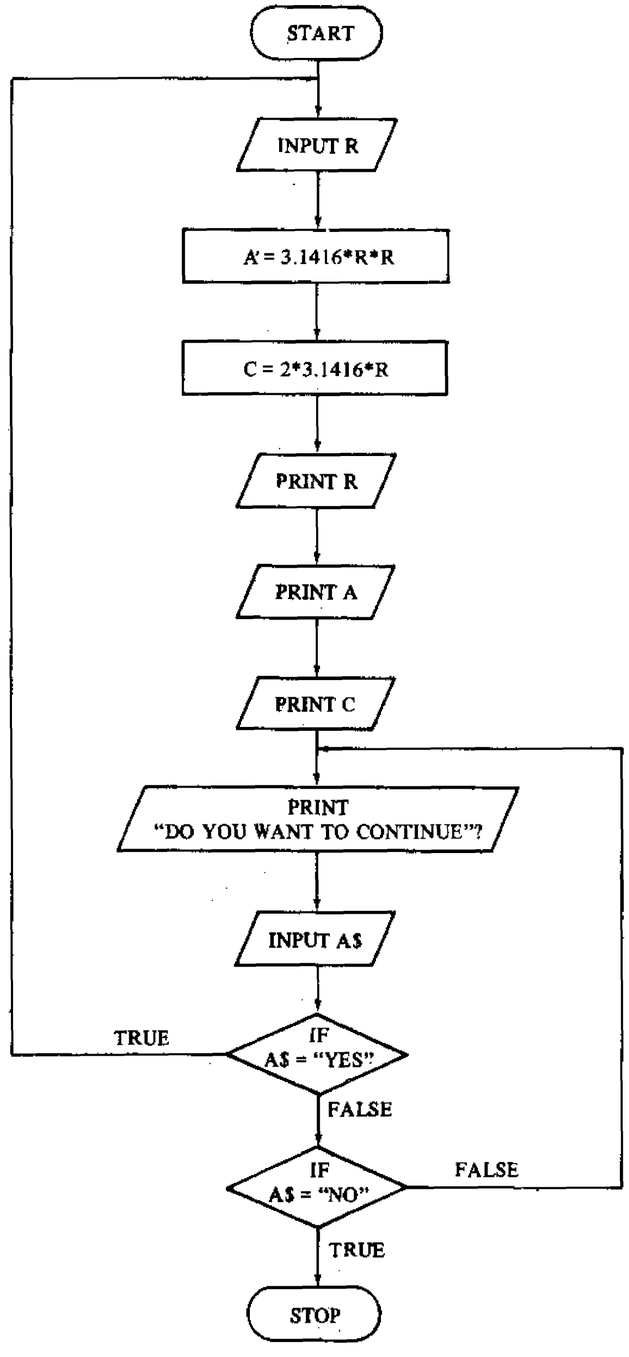


FIG. 3.4

MORE ABOUT PROGRAMMING

In the last chapter, we introduced the basic computer programming concepts. Now we are ready to move into more advanced and useful areas.

Did you ever play games with a computer? If yes, do you know how the computer shuffles cards when you play Blackjack; or how to roll dice to determine your next move in craps?

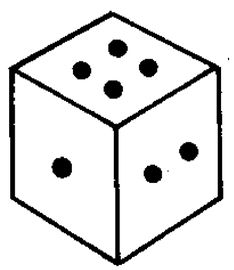
In the System 80, there is a built-in function – RND (n) – that generates random numbers.

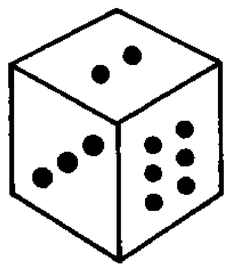
(Note: n is the upper bound of the numbers to be generated. Please refer to the built-in functions in the Basic Manual).

With the RND(n) function, the computer can automatically and randomly shuffle cards, roll die, etc. Furthermore, in order to ensure the numbers generated are at random after each power up, the function RANDOM should be used.

Example

```
10 RANDOM
20 N = RND (6) :PRINT
30 PRINT "THE OUTCOME IS:"; N
40 INPUT "DO YOU WANT TO CONTINUE (YES/NO)"; A$
50 IF A$ = "YES" OR A$ = "Y" GOTO 20
60 IF A$ = "NO" OR A$ = "N" THEN END
70 GOTO 40
READY
>RUN
THE OUTCOME IS: 4
DO YOU WANT TO CONTINUE (YES/NO) ? YES
THE OUTCOME IS: 4
DO YOU WANT TO CONTINUE (YES/NO) ? Y
```





```
THE OUTCOME IS: 6
DO YOU WANT TO CONTINUE (YES/NO) ? NN
DO YOU WANT TO CONTINUE (YES/NO) ? NO
```

The flow chart for this program should be:

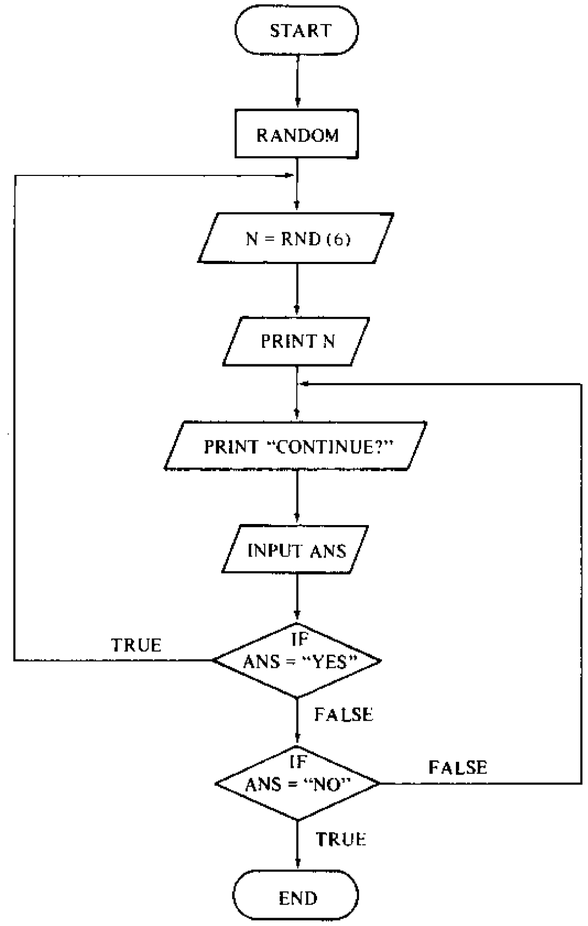


FIG. 3.5

The program generates a random number from 1 to 6 (simulating the faces of die), prints out the number, then asks the operator whether he or she wants to continue. If the input is "YES" or "Y" then the computer repeats the process, if the input is "NO" or "N" then the program ends. However, if the input is neither "YES" nor "NO", the computer, looks for a valid and acceptable response, asks repeatedly if the player wants to continue.

Besides of the random number function, there are many many useful built-in functions that help you to solve a wide range of problems simply by using a function call.

One more example

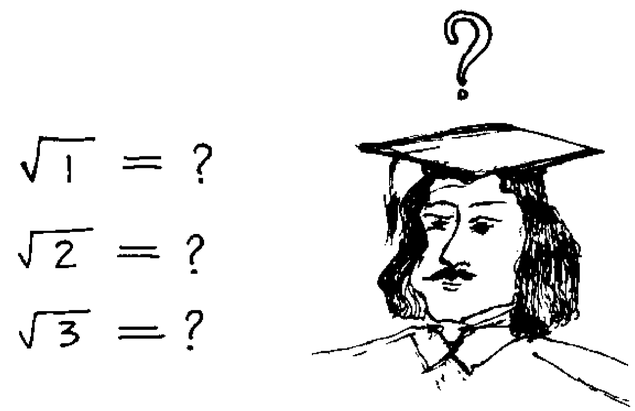
```
10 FOR A = 1 TO 10 STEP 1
20 PRINT A, SQR (A)
30 NEXT A
40 END
```

This program prints out the square root of the values from 1 to 10. The initial value of A is 1; then in line 20, 1 and the square root of 1 are printed. A is incremented by 1 in line 30, and the computer checks if A is equal to 10, if not, go back to line 10. The sequence continues until A is equal to 10.

From time to time, you may encounter strings (series of numbers and characters) instead ot just figures. In other words, you need some knowledge in string manipulation.

For example, you are handling the shipping job of a company which is using code names that contain both letters and numbers for its orders.

The format is: AB1234S001
ITEM NO. STORE NO



The first 6 characters are the item number code, while the last 4 characters are the store number code.

Suppose there are 20 stores. Stores no. 1 to 7 located in city A; stores no. 8 to 20 located in city B. Therefore, it is profitable to pool the orders in store no. 1 to 7 together, and store no. 8 to 20 together for cheaper shipping charges.

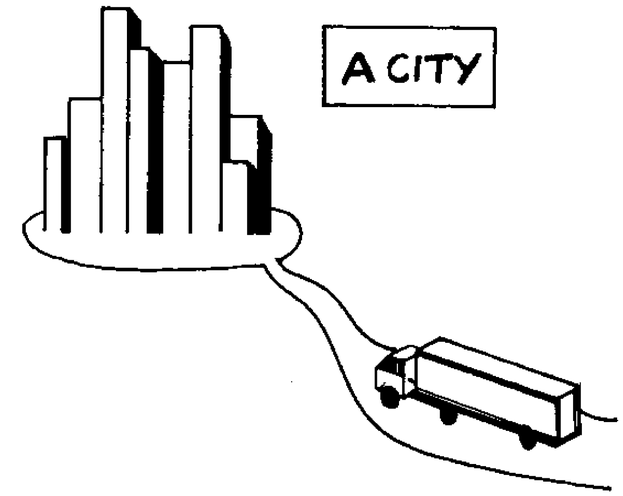
Now what we are going to do is using the computer to check if the store no. is less than eight, then we add the orders to the shipment to city A. Otherwise, we add the orders to the shipment to city B.

Try to follow this program.

```
10 INPUT "ENTER THE CODE NAME";CN$
20 IN$ = LEFT$ (CN$,6)      .REM  FIRST 6 LETTERS = ITEM NO.
30 SN$ = RIGHT$ (CN$,3)     .REM  LAST 3 LETTERS = STORE NO.
40 SN = VAL (SN$)           .REM  CHANGE STRING VALUE TO ARITH. VALUE
50 INPUT "ENTER THE ORDER QUANTITY";Q
60 IF SN > 8 THEN GOSUB 1000 ELSE GOSUB 2000
•
•
```

This is not a complete program, because you need some operations (program statements) in subroutines 1000 and 2000

Note: please refer to the BASIC manual for the usage of LEFT$, RIGHT$, and VAL functions.



Now, are you ready to step into arrays? It is an interesting section, don't miss it.

Suppose we have a set of 24 numbers.

100,	110,	120,	130,	140,	150,	160,	170,	180,	190,	200,
210,	220,	230,	240,	250,	260,	270,	280,	290,	300,	310,
320,	330,									

There are many ways we can refer to a particular number in this set. For example, we can say the 18th one is 270 which can also be referred as the 7th one on row 2.

In the case of a computer, we can assign each of these numbers to a variable. Such as:

```
10 A1 = 100
20 A2 = 110
30 A3 = 120
40 A4 = 130
50 A5 = 140
•
•
•
```

So that we can reference each of them easily!
But if we have more numbers, we may run out of variables.
A better way to handle a set of numbers or strings is to use arrays.

For these 24 numbers, we may use a single dimension array:

```
10 CLEAR 200      :REM  CLEAR 200 BYTES OF MEMORY
20 DIM A(23)      :REM  A 24 ELEMENT ARRAY A( 0 TO 23)
```

In a program, we may assign:

```
10 A(0) = 100
20 A(1) = 110
30 A(2) = 120
```

In order to reference 270 in the number list, we may just say

```
50 PRINT A (17)    :REM  PRINT THE 18TH ELEMENT OF THE ARRAY
100 A(17) = A(17) + 100    :REM  ADD 100 TO THE 18TH ELEMENT
```

Isn't it simple?
If we divide these 24 numbers into 3 groups, with 8 elements in each group. We may use a two dimensional array:

```
20 DIM B (2,7)    :REM  A 3 X 8 TWO DIMENSIONAL ARRAY
```

Then the numbers may be arranged like:

ROW	0	100	110	120	130	140	150	160	170
ROW	1	180	190	200	210	220	230	240	250
ROW	2	260	270	280	290	300	310	320	330
COLUMN		0	1	2	3	4	5	6	7

This time if we refer to 270, we should say:

```
100 PRINT B(2,1)    :REM PRINT THE ELEMENT AT ROW 2, COLUMN 1
230 F = B(2,1) + 10  :REM  ADD 10 TO THE ELEMENT
```

Similarily, if we refer to 160, we should say

```
210 PRINT B(0,6)     :REM PRINT THE ELEMENT AT ROW 0, COLUMN 6
```

Let us go one step further, with these 24 numbers, we may divide them into even smaller groups. Say 4 pages of 2 x 3 arrays. In other words, that is a three dimensional array:

```
20 DIM E(3,1,2)     :REM  A 24 ELEMENTS ARRAY E ( 4 X 2 X 3 )
```

PAGE 0

ROW 0	100	110	120
ROW 1	130	140	150
COLUMN 0	1	2	

PAGE 1

160	170	180
190	200	210

PAGE 2

220	230	240
250	260	270

PAGE 3

280	290	300
310	320	330

This time, if we refer to 270 again, we should say

```
70 PRINT E(2,1,2)     :REM  PRINT THE ELEMENT ON PAGE 2, ROW 1, COLOMN 2
```

If we refer to 200, we should mention E (1, 1, 1).

Try to follow this program:

```
10 DIM E(3,1,2)          :REM  E IS A 4X2X3 ARRAY
20 FOR P = 0 TO 3        :REM  LOOP TO NEXT P 4 TIMES
30 FOR R = 0 TO 1        :REM  LOOP TO NEXT R 2 TIMES
40 FOR C = 0 TO 2        :REM  LOOP TO NEXT C 3 TIMES
50 INPUT "ENTER A NUMBER"; E(P,R,C)    :REM  INPUT AN ELEMENT
60 NEXT C
70 NEXT R
80 NEXT P
90 FOR P = 0 TO 3
100 FOR R = 0 TO 1
110 FOR C = 0 TO 2
120 PRINT E(P,R,C);      :REM  PRINT EACH ELEMENT
130 NEXT C
140 PRINT
150 NEXT R
160 PRINT
170 NEXT P
180 END
```

This program prints those 4 pages of numbers, if the inputs are in the correct sequence.

This is the basic concept of arrays, now you can do your own programming; the only limitation is your imagination, not your computer.