CHAPTER 6
BEGINNING BASIC PROGRAMMING

- Introduction
- Programming modes
- Input/Output statements
- Control statements and loops
- Conditional statements
- Subroutines
- REMarks

INTRODUCTION

Up until now, you may or may not have understood exactly what was going on in the programs that introduced you to some of the capabilities of your Plus/4. This chapter will explain some of the BASIC commands that were used in those programs. This chapter focuses on some of the more often used BASIC statements that you will need to construct your own programs. At the end of this chapter we will touch on some programming techniques. This chapter will give you a quick introduction to programming, but it is still an introduction. To really learn to program, we suggest you pick up a good book on BASIC at your local bookstore. (See the bibliography in Section 14 of the Encyclopedia for suggestions.) There are many versions of BASIC, each a little different. Your Plus/4 is equipped with an advanced version of the BASIC language called Commodore BASIC 3.5.

PROGRAMMING MODES

Your Plus/4 gives you two ways to use BASIC statements and commands: in direct mode and indirect mode. Direct mode is often referred to as immediate mode, and indirect mode is also known as program mode.

DIRECT, or IMMEDIATE MODE, as the name implies, executes statements and commands immediately (as soon as you press RETURN after typing in a command). You do not type line numbers when using commands or statements in direct mode. You only type the command or statement and press the RETURN key. This mode is used if you want your computer to perform calculations and give you an immediate result. Commands such as LIST, SAVE, LOAD, VERIFY and RUN are usually used in direct mode. Most (but not all) BASIC statements work in direct mode.

INDIRECT, or PROGRAM MODE, allows you to organize a series of BASIC statements into a set of instructions that will be performed in the order that you decide. Each of the lines in the program has a line number which tells the computer to execute the lines in a certain order. You've already seen several examples of program mode in Chapter 4. Remember that when you use program mode, you must press RETURN to enter each line of the program into the memory of your Plus/4. If you don't press RETURN , and just go to the next line, the line you typed has not been entered. Once the program is in memory, nothing will happen until you enter the RUN command. The RUN command tells the Plus/4 to execute the program starting with the lowest numbered line.

Lines are most often numbered by tens, since you'll frequently have to add lines in different places in the process of writing a program. You could, if need be, add nine new lines between line 10 and line 20 in a program. However, your Plus/4 features a BASIC command, RENUMBER, that allows you to add new lines and change the existing line numbers.

This saves a lot of confusion that often occurs when changing and rearranging lines.

INPUT/OUTPUT STATEMENTS

Input/Output (I/O) statements are used in programs to communicate with the person RUNning the program. Before the program is run, if all the data for the calculations is available, there is really little need for input statements. It is often more useful if the computer can get data from the person RUNning the program (we'll call him or her the program user). Programs are much more versatile if the data is not "set in stone" before running them. Output statements can be used by the computer to tell the person running the program the answers that the computer has calculated. Obviously, output statements are vital; there would be little sense in RUNning a program that had no output statements. (Kind of like a tree falling in a forest with no one around to hear; does it make a sound? Does it matter?)

Advanced programmers also use I/O statements to communicate with devices instead of with the program user. You've probably done this yourself, but not in a program—when you used LOAD or SAVE with your Datassette or Disk drive. LOAD is basically an input statement since the Plus/4 gets data (your program) from your Datassette or disk drive while SAVE is an output statement, as the Plus/4 sends data to those devices.

In this introduction to I/O statements, we will limit ourselves to a few of the most important ones, the ones that you'll need immediately. They are: PRINT, INPUT, GETKEY, and READ/DATA. PRINT is an output statement, while the others are input statements. (Remember that all BASIC I/O statements can be found in the BASIC Encyclopedia at the end of this book.)

Statement name: **PRINT**

Format: **PRINT** "text in quotes" or variables or numbers or calculations, etc.

You have used the PRINT statement often in programs in earlier chapters. From that, and from the format example above, you can see that PRINT is a very versatile statement. You can use it to PRINT out messages, pictures made out of graphic characters, perform calculations, display the value of a variable, and more. Since the PRINT statement is used so often, it pays to learn to use it well.

Use #1 Text Display

Suppose in your program, you want to inform the user that his or her checking account balance is negative, or that pur-

ple lizards are not allowed in the control room. The easiest way would be to PRINT your statement as a text string. Text strings are printed out exactly as you type them in. They must be surrounded by double quotes (" "). For example:

```
100 PRINT "YOU ARE BROKE!"
```

would tell the user that there is no money left, while

```
150 PRINT "YOU CAN'T BRING YOUR FRIEND INTO
THE CONTROL ROOM"
```

could be used in the second example.

Whatever appears between the quotation marks is known as a literal, because it is PRINTed exactly as it appears. It doesn't matter whether it is words, letters, numbers, punctuation marks, etc.

Certain keys, like the cursor and color keys, act differently when used in text strings. Instead of changing the color or moving the cursor when you type the key, a reverse character is printed in the string. When the program is RUN, the character is translated into what you wanted typed in the first place. This lets you clear the screen, change the color you are PRINTing in, move the cursor, all within your program. For example, try this:

```
10 PRINT " SHIFT CLR/HOME CONTROL 3
TESTING, CRSR-DOWN CONTROL 7 TESTING"
```

Remember to type the following keys simultaneously when you use the SHIFT and CONTROL keys. The reversed symbols are the signals to the computer that tells it to perform the clear screen, color change or move the cursor.

Use #2 Printing Numbers and Calculations

PRINT can display the answer to a calculation made within the print statement. (SEE NUMBERS and CALCULATIONS). The Plus/4 performs the operations needed to get the answer, then displays it on the screen. For example:

```
100 PRINT 58* 15,23,45+ 1000-45* (4-3)
    prints:
 870   23   1000
```

This gets more interesting when variables are also used. User input can be displayed, and earlier calculations saved in variables can also be PRINTed out, or even used in additional calculations.

Examples:

TYPE:

```
10 R= 10 * 2: N= R– 5
20 PRINT "R IS ";R;" AND N IS ";N
30 PRINT "BUT R TIMES 2 IS";R*2
40 PRINT "AND N MINUS 2 IS";N– 2
```

Normally, after each PRINT statement, the cursor automatically goes to the beginning of the next line. You can override this by putting a semicolon (;) after the PRINT statement like this:

```
200 PRINT "THESE TWO SENTENCE PARTS WILL BE ";
210 PRINT "PRINTED ON THE SAME LINE"
```

Statement name: **INPUT**

Format: **INPUT** "optional message";variable to be input

The INPUT statement lets you get data from the program user through the keyboard, and use it in the program. The optional message lets you tell the user exactly what you are asking for; the message is printed when the INPUT statement is executed, along with a question mark. Then the Plus/4 waits for the user to type an answer, followed by pressing the RETURN key. The input from the user is placed in a variable. You can either get a string from the user by using a string variable (A$, for example), or a number by using a numeric variable. The INPUT statement can only be used in program mode.

Examples:

TYPE:

```
10 PRINT "WHAT IS YOUR NAME";
20 INPUT A$
30 PRINT "I AM PLEASED TO MEET YOU";A$;"."
40 INPUT"HOW OLD ARE YOU";AG
50 PRINT AG;" IS A BIT OLDER THAN I AM."
RUN
```

Statement name: **GETKEY**

Format: **GETKEY** variable to be input

GETKEY is another way for you to enter data while the program is being RUN. The GETKEY statement accepts only one key at a time. Whatever key is pressed is assigned to the

string variable you specified in the GET statement (A$, for example). GETKEY is useful because it allows you to enter data one character at a time without having to press the RETURN key after each character. The GETKEY statement may only be used in a program.

Example of GETKEY in a program:

```
1000 PRINT "PLEASE CHOOSE A, B, C, D, E, OR F"
1010 GETKEY A$
```

Statement Name: **READ/DATA**

Format: **READ** variables to be input
DATA data items to be read

The READ/DATA statements are used as a convenient way to assign values to variables. You can think of the READ statement as an INPUT statement that asks the Plus/4 for the data, rather than the user. The data is (naturally enough) kept in DATA statements. When the Plus/4 executes a READ statement, it looks at the next data item in the DATA statement, and assigns it to the variable in the READ statement.

The READ statement is always used with a DATA statement. A DATA statement is just a line of data (words or numbers) in a program. The READ statement is used to assign those values to variables. (For each variable listed in the READ statement, your Plus/4 "reads" a value from the DATA line for that variable.) A DATA statement is not executable and can appear anywhere in the program. The thing to remember about the READ statement is that the variable type must be the same as the type of data available in the DATA statement (number variables for numbers, text variables for text). Otherwise, a TYPE MISMATCH ERROR occurs.

Example:

```
10 READ A$,B$,C$,D$,E$
20 PRINT A$:PRINT B$:PRINT C$
30 PRINT D$: PRINT E$
40 DATA GROUCHO, HARPO, CHICO
50 DATA ZEPPO, GUMMO
```

The computer responds with:

```
GROUCHO
HARPO
CHICO
ZEPPO
GUMMO
```

CONTROL STATEMENTS AND LOOPS

It would be pretty boring if your computer could only execute program lines in order. The computer could only start at the beginning and go through each step in order until the end of the program. This would lead to very long programs; if you wanted to do the same thing twice (like PRINT "HELLO"), you would have to duplicate the program lines. With a small example like PRINTing HELLO, this doesn't make a lot of difference, but it could become difficult in larger programs. This is why computers have control statements. Control statements tell the computer to ignore the normal order of the program lines, and go to another line regardless of the sequence. The Plus/4 has several varieties of control statements: unconditional (like GOTO) which *always* transfer control; counting statements (like FOR/NEXT) which transfer control a specified number of times; and, for you structured programming fans out there, DO/LOOP.

Statement Name: **GOTO**

Format: **GOTO** line #

GOTO tells your computer to immediately go from the current line in your program to the line number specified in the GOTO statement. For example, if line 20 reads GOTO 40, your Plus/4 would jump to line 40, skipping any statements between 20 and 40.

Example using GOTO statement in a program:

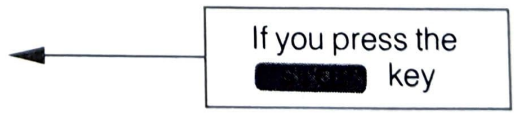
TYPE:

```
10 PRINT "A PENNY SAVED IS BETTER THAN
NOTHING"
20 GOTO 10
```

The computer responds by printing the message in line 10 again and again, until you press the STOP key, like this:

```
A PENNY SAVED IS BETTER THAN NOTHING
A PENNY SAVED IS BETTER THAN NOTHING
A PENNY SAVED IS BETTER THAN NOTHING
```

```
BREAK IN 10
READY.
```



This print statement will continue 'forever'. Every time your Plus/4 gets to the GOTO in line 20 it goes back to line 10. This is called an INFINITE LOOP in computerese. While you might want to do this, usually you want to repeat only a certain number of times, or until something happens. That is

why the FOR/NEXT and DO/LOOP statements are available in BASIC.

GOTO can also be used in direct mode. GOTO line # will start the program at the line you specify, while keeping the variables the same (instead of clearing them as RUN does).

Statement Name: **FOR/NEXT**

Format: **FOR** variable = start value **TO** end value

some BASIC statements

NEXT variable

The FOR/NEXT statements let you create a loop that will repeat a certain number of times. The program statements between the FOR statement and the matching NEXT statement are repeated in the loop. The variable in the FOR statement acts as a counter. It is initially set at the start value you supply. Then, the program lines after the FOR are executed, until the computer gets to the matching NEXT statement. The NEXT tells your Plus/4 to add one to the counter. If the counter is less than or equal to the end value, the computer returns to the program line after the FOR statement. Otherwise, your Plus/4 continues with the first statement after the NEXT.

Example using a FOR/NEXT loop

```
10 PRINT "COUNTUP..."
20 FOR J= 1 TO 10
30 PRINT "WE HAVE";J
40 NEXT J
50 PRINT "WE COUNTED UP TO";J
```

One more thing about FOR/NEXT: you can also specify a STEP value in the FOR statement. Instead of adding 1 to the counter variable, your Plus/4 adds your STEP value. If you use a STEP of 5 with the statement FOR M = 10 TO 30, for example, the counter would count 10, 15, 20, 25, 30 after each loop. The STEP command even lets you count backwards (by using a negative STEP value).

Another example, with a negative STEP:

```
10 PRINT "COUNTDOWN..."
20 FOR J= 10 TO 1 STEP -1
30 PRINT "WE ARE AT";J
40 NEXT J
50 PRINT "WE HAVE LIFT-OFF AT";J
```

Statement Name: **DO UNTIL/WHILE...LOOP UNTIL/WHILE**

Format: **DO UNTIL** [condition] | **WHILE** [condition]

some BASIC statements

[**EXIT**]

LOOP UNTIL [condition] | **WHILE** [condition]

The DO/LOOP statement combination is another way to create a loop. This statement combination is very powerful and versatile. The DO/LOOP method of loops is a common technique of structured programming languages. In this chapter we'll discuss just a few possible uses.

If you want to create an infinite loop, just start a section of program lines with DO, and end it with a LOOP statement, like this:

```
100 DO: PRINT "GOING UP"
110 LOOP
```

Press the STOP key to end the program.

A more useful form is to combine the DO/LOOP with the UNTIL statement. The loop will run continually unless the condition for UNTIL happens.

```
100 DO: INPUT "DO YOU LIKE YOUR COMPUTER";A$
110 LOOP UNTIL A$= "YES"
120 PRINT "THANK YOU"
```

For the other ways you can use the DO/LOOP, see the BASIC Encyclopedia at the end of this book.

CONDITIONAL OR DECISION MAKING STATEMENTS

Conditional statements are used to make decisions. One of the most powerful abilities of a computer is to make decisions based on what is going on. One of the conditional statements available on the Plus/4 is known as IF/THEN statements.

Statement Name: **IF/THEN**

Format: **IF** condition **THEN** do this (only if the condition is true)
Basically, the IF/THEN statement works like this:

IF (this statement is true) THEN (do this statement)
Actually, you have always known how conditional statements work. How many times have you heard this famous line?:

IF you eat all your vegetables THEN you can have dessert. That may seem a bit trivial, but that is the gist of the IF/THEN statement.

If the condition in the IF statement is true, everything after the THEN is executed.

EXAMPLE:

```
10 INPUT"WHAT'S THE TENTH LETTER OF THE
ALPHABET";A$
20 IF A$= "J" THEN PRINT "RIGHT": GOTO 100
30 INPUT "IS THIS AN A";X$
40 IF X$= " YES " "LOOP" THEN 60
50 PRINT "WRONG, TRY AGAIN": GOTO 30
60 PRINT "TYPE A B"
70 GETKEY A$:IF A$= "B"THEN PRINT "RIGHT"
100 PRINT "THAT'S ENOUGH OF THIS, ANYWAY"
```

In line 40 , we just say THEN 60. This actually means THEN GOTO 60, but since the THEN GOTO combination is used so often, BASIC allows you to leave off the GOTO. An optional step for the IF/THEN statement is the ELSE clause, that directs your computer to a specific action if the original IF condition was not met. An example showing the ELSE clause would be: IF B > 5 THEN 40 ELSE GOTO 10. The BASIC Encyclopedia explains the IF/THEN/ELSE statement more fully.

SUBROUTINES

If you have something in your program that has to be repeated in more than one place in your program, you have two choices: you can have duplicate routines, or you can create a subroutine. A subroutine is a section of your program that can be used from anywhere else in your program. When the subroutine is finished, the program automatically continues at the statement just after where the subroutine was called.

Statement Name: **GOSUB/RETURN**

Format: **GOSUB** line #

The GOSUB statement is used to call a subroutine. Like the GOTO statement, control is transferred to the line number specified in the statement. However, unlike the GOTO, the Plus/4 remembers where the GOSUB is located. When a RETURN is next encountered, control returns to just after the GOSUB statement.

Example:

```
5 T= 0:FOR J = 1 TO 99
10 PRINT "GIVE ME A NUMBER FROM 1 TO 10"
20 INPUT N
30 IF N<1 THEN GOSUB 100:GOTO 20
40 IF N>10 THEN GOSUB 100:GOTO 20
50 T= T+ N
60 NEXT J
70 PRINT "THE TOTAL IS"T
80 END
100 PRINT "THAT NUMBER IS OUT OF RANGE"
105 PRINT "PLEASE TYPE A NUMBER BETWEEN
1 AND 10"
110 RETURN
```

If a RETURN is encountered when there are no active GOSUBs, you get a RETURN WITHOUT GOSUB ERROR. You should be careful that the computer never gets into one of your subroutines except by GOTO. One method is to group the GOSUB and GOTO statements together, protected from normal program execution by an END statement.

REM STATEMENTS

Statement Name: **REM**

Format: **REM** message

The REM statement is used to comment (or REMark) on your programs. The REM statement is not executed as part of the program; it is a message that can be seen only when looking over the LISTing of a program. Often, if you don't comment, six months after you write the program you might forget what some part does. You can use REM statements to put in reminders, so you can more easily figure out what you really meant, or give others information with your messages.

Example:

```
1560 E= R/I* 9:REM THIS FIGURES OUT A
PITCHER'S ERA
100 INPUT A, B: REM A IS HEIGHT IN INCHES AND B IS
WEIGHT
```

SUMMARY

As we REMarked in the introduction, this would not be a complete tutorial on BASIC. We just gave you some of the BASICs. Every BASIC command in the Plus/4 is in the BASIC Encyclopedia, with format, description, and examples. Don't be afraid to experiment. If you are serious about learning BASIC, get some of the books on BASIC programming listed in the Section 14 of the Encyclopedia. Programming is like eating salted peanuts: once you start, you may not be able to stop.