values 8 for cursor left, 28 for cursor right, 29 for cursor up and 30 for cursor down. Your computer will probably use different values. Substituting the values you have found for your computer's cursor control codes in the program above, try the following program:

```
10 PRINT CHR$(12): REM USE CLS OR
   APPROPRIATE CODE
20 FOR L = 1 TO 39
30 PRINT "*";
40 NEXT L
50 FOR L = 1 TO 22
60 PRINT CHR$(8); :REM USE 'CURSOR LEFT' CODE
70 NEXT L
80 FOR L = 1 TO 4
90 PRINT "@";
100 NEXT L
110 END
```

This should print a line on the screen looking like:

```
**********@@@@**********
```

Lines 20 to 40 would simply have printed a line of 39 stars. However, lines 50 to 70 'printed' the cursor left 'character' 22 times, so the cursor moved back along the line 22 places. Lines 80 to 100 then printed @ four times and the program then ended. Programming techniques such as this allow the programmer to move the cursor around the screen to print new characters in new positions that may not be known until the values are calculated in the program. This technique has the advantage of enabling ordinary screen characters to be used to plot simple graphs, without resorting to the computer's special graphics facilities (if it has any).

To see how this kind of cursor control can be used to produce graphs as an output from your programs, try the following short program:

```
10 PRINT "THIS PROGRAM PRINTS A BAR GRAPH OF
   3 VARIABLES"
20 INPUT "INPUT THE THREE VALUES ";X,Y,Z
30 PRINT
40 FOR L = 1 TO 2
50 FOR A = 1 TO X
60 PRINT "*";
70 NEXT A
80 PRINT CHR$(13)
90 NEXT L
100 FOR L = 1 TO 2
110 FOR A = 1 TO Y
120 PRINT "+";
130 NEXT A
140 PRINT CHR$(13)
150 NEXT L
160 FOR L = 1 TO 2
170 FOR A = 1 TO Z
180 PRINT "#";
190 NEXT A
200 PRINT CHR$(13)
210 NEXT L
220 PRINT
230 END
```

The program prints out a bar graph of the three variables. The bars are printed in horizontal rows, starting from the left and following the 'natural' cursor movement. Notice that a PRINT CHR$(13) is needed in lines 80, 140 and 200. They are needed because semi-colons at the end of PRINT statements suppress carriage returns (13 is the ASCII code for <CR>).

More About Variables

So far we have treated variables as though there were only two kinds (numeric and string). In fact, there are several types of numeric variables recognised by BASIC, and a good programmer will always specify the right type to economise on memory and ensure correctness.

When a variable is declared in a programming language, a certain amount of memory space will be automatically allocated to store that variable. If the program knows that the variable will always be an integer, (e.g. LET SCORE = TOTAL + BONUS − PENALTY) less memory needs to be set aside for the variable. If we have a variable that can take an infinite number of different values (e.g. LET AREA = PI * RADIUS * RADIUS), more memory space will have to be allocated.

In the development of our computerised address book, we became familiar with the convention of specifying string variables by using the $ sign after the variable name (e.g. LET SCHKEY$ = MODFLD$(SIZE)). Variables without the 'dollar' sign were assumed to be ordinary numeric variables. However, similar conventions can be used after variable names to specify the type of numeric variable. A variable name with no specifier is assumed to be a real numeric variable of single precision. Other signs recognised by most BASICS include: % to specify an integer variable, ! to specify a single precision variable, and # to specify a double precision variable (i.e. the variable can store twice as many significant digits). Here is a fragment of a hypothetical program that uses these signs:

```
70 LET PLAYER$ = "JOHN": REM A STRING
   VARIABLE
80 LET SCORE% = 0: REM AN INTEGER VARIABLE
90 LET PI! = 3.1416: REM A SINGLE PRECISION
   VARIABLE
100 LET AREA# = PI*R*R: REM DOUBLE PRECISION
    VARIABLE
110 LET GOES = 6: REM ASSUMED TO BE SINGLE
    PRECISION REAL
```

Having said that, it must be pointed out that not all BASICS support all these variable types. The Spectrum, for example, does not have integer variables. Integers are simply stored as single precision real numbers. Neither does it support double precision numbers. However, single precision numbers in Spectrum BASIC are calculated to nine significant figures, against only seven significant figures in Microsoft BASIC. The BBC Micro does support variables of the integer type and single precision reals calculated to nine