CHAPTER 1
Simple Functions?

Displaying numbers neatly

A whole number or a number without any decimal part is called an integer. Displaying integers neatly on your screen is straightforward - simply use the following line:

```
PRINT TAB(25) X
```

The TAB function simply moves the cursor to the appropriate position on the screen. For example, the numbers 9, 123, -10 and 89 would appear as follows:

```
   9
123
-10
 89
```

For non-integral numbers the display goes astray: the numbers 89, 1.2, -13.89 and 0.12 would appear as:

```
      89
     1.2
-13.89
   0.12
```

The numbers are right-justified; but it would be nice to have the decimal points vertically aligned. Format commands may be used to achieve this: the command @% = &02020A vertically aligns the numbers and prints to two decimal places. The above four numbers would now appear as:

```
  89.00
   1.20
-13.89
        0.12
```

The format command @% = &02030A would display the above numbers

as:

```
  89.000
   1.200
-13.890
   0.120
```

To allow for a variable number of decimal places and to avoid printing the redundant zeros after the decimal point, requires an alternative method. We can use the function INT(X) and ABS(X).

The function INT(X) returns the integral part of X, that is, the largest integer which is less than or equal to X. For example

```
INT(1.21) = 1
INT(2) = 2
INT(2.1) = 2
INT(-2) = -2
INT(-2.1) = -3
INT(9.1) = 9
INT(-9.2) = -10
```

The function ABS(X) returns the absolute value of X, that is, the number ignoring the + or - sign, eg:

```
ABS(9.1) = 9.1
ABS(-9.1) = 9.1
```

The following program lines will display numbers neatly on your screen:

```
@% = 10 (set format to default value)
Y = INT(ABS(X)) : L = LEN(STR$(Y))
IF X < 0 THEN L = L + 1
PRINT TAB(25-L);X
```

A typical display is shown below:

```
        3
        0.23
     -89.14
6712399.1
        2.23871
       -1.22
       -0.13
```

In the first line of the program, ABS takes care of negative numbers while INT takes care of non-integral numbers. The function STR$(Y) converts the number Y into a string, LEN calculates its length and TAB moves the cursor to the appropriate position on the screen. The second line in the program is needed to take care of negative numbers.

Note that using the INT function without the ABS function will not work (for example look at the number -9.1); even if the second line is removed (try -8.1 and -9.1).

The program above illustrates one simple use of the functions INT and ABS. It works except for numbers which are close to 0 (absolute value less than or equal to 0.01) or which are very large (absolute value greater or equal to 2147483647). In fact whenever the scientific notification E appears the display goes slightly astray.

```
       3
-89.14
 1E-04
```

You might like to add two lines to our program to take care of numbers involving scientific notation E.

Rounding off numbers

The INTegral function is useful for 'rounding off' numbers. For instance if you had £565.58 in a bank account and received 9% interest per annum then the amount you expect to have after one year is

```
564.58 + 565.58*9/100
```

Using your BBC or Electron you can check that this has a value of 616.4822. But, of course, the bank would 'round' this DOWN to £616.48. Similarly an amount such as 76.6752 would be rounded UP to £76.68. Your computer can do this rounding off with the following line.

```
X = INT(X*100 + 0.5)/100
```

Here X is first multiplied by 100 to convert to pence. Then 0.5 is added which causes a rounding up if the fraction of pence is greater or equal to one-half. The INT function ignores any decimal parts and finally the number is divided by 100 to convert it back into pounds.

In general, the program line

```
B = INT(A*10^D + 0.5)/10^D
```

gives the value of A rounded off to D decimal places.

You could use a format command to simulate a rounding off - but this may lead to problems. Try the following

```
@% = &02020A
X = 3.0051
PRINT X
```

The resulting number printed is 3.01 which is 3.0051 rounded off. However, try this:

```
@% = &02020A
X = 3.0051 : Y = 3.0051
PRINT X, Y, X+Y
```

The numbers printed are 3.01, 3.01 and 6.01: the last number is not the sum of X rounded off with Y rounded off, instead it is X + Y rounded off.

Bank balances

The ideas given earlier on this chapter for displaying numbers neatly on the screen could be used, for example, in a bank balance program. Using the rounding off ideas and the format command @% = &02020A, you may end up with a display such as:

DETAILS	PAYMENTS	RECEIPTS	BALANCE
B/F			596.61
869162	46.22		550.39
869164	169.00		381.39
869165	15.01		366.38
CHQS		75.70	442.08

Overdrawn bank balances

Bank balances occasionally become overdrawn (or go into the red). This occurs when the balance becomes negative (less than zero). Thus a balance of -£64.00 means that you are overdrawn by £64.00. By using a line such as the following, you could indicate when the amount shown is overdrawn.

```
PRINT X;: IF X<0 THEN PRINT;"DR"
```

For example:

DETAILS	PAYMENTS	RECEIPTS	BALANCE
B/F			442.08
869166			389.29
869167	422.00		32.72 DR

Colourful balances

The function SGN(X) is the sign function which returns the sign (positive, negative, or zero) of the number X. The result is +1 if the number is positive, -1 if it is negative, and 0 if it is zero. For example:

```
SGN(9.21) = 1
SGN(-9.1) = -1
SGN(0) = 0
```

A typical use of the SGN function is when the program is required to perform different subroutines depending upon whether the sign of a number is positive, negative, or zero. For example, the program line

```
ON SGN(X) + 2 GOSUB 1000, 1100, 1200
```

would cause the program to execute the subroutine 1000 if X is negative, subroutine 1100 if X is 0, and subroutine 1200 if X is positive.

An interesting use of the SGN function is a simple method of changing the colour of printing. On the BBC micro in MODE 7, CHR$(129) represents red, CHR$(130) represents green while CHR$(131) represents yellow. Thus CHR$(130+SGN(X)) will be red, green or yellow depending on whether X is negative, zero or positive. Furthermore, CHR$(137) causes the printing to flash while CHR$(136) and CHR$(138) are steady. Thus CHR$(137 + SGN(X)) will print flashing or steady depending on the sign of X. Try the following program.

```
   10 MODE 7
   20 INPUT "Number",X
   30 PRINT'CHR$(130+SGN(X))CHR$(137+SGN
(X))ABS(X)
   40 GOTO 20
```

You can produce similar results in other modes on the BBC micro and on the Electron, by using statements like COLOUR 2 + SGN(X), etc.

```
   10 MODE 2
```

```
    20 INPUT "Number",X
    30 COLOUR 2 + SGN(X):PRINT'ABS(X):CO
LOUR 7
    40 GOTO 20
```