easier to enter them using a READ statement together with a DATA statement, like this:

```
10 DIM A(5)
20 FOR X = 1 TO 5
30 READ A(X)
40 NEXT X
50 DATA 5, 10, 15, 20, 100
```

Try this short program, and then test the contents of the array using the PRINT command (that is, use PRINT after the program has been RUN. For example, PRINT A(1)‹CR› and PRINT A(5). Now we can add a few lines to the program to print the elements in the array for us automatically:

```
60 FOR L = 1 TO 5
70 PRINT A(L)
80 NEXT L
90 END
```

RUN this program and check that the correct values are printed on the screen. Then retype line 50 using five different DATA items. Remember that the numbers in a DATA statement must be separated from each other using commas, but there must be no comma before the first number or after the last one.

The simplest way to assign values is to use READ and DATA statements. If the values will be different every time the program is run, using the INPUT statement inside a FOR-NEXT loop is probably the best way. If the total number of elements in the array is fixed, the number can be used as the upper limit in the FOR statement.

Let's use all we have learnt so far to build a short but powerful program. Suppose we wanted to sort some numbers into ascending order. Before setting out to write the program, the first thing to do is to figure out how to solve the problem in a logical way. When the way to solve the problem seems clear, write down the steps one after the other using clear, short English sentences.

Suppose we start with five numbers: 4, 9, 2, 8, 3. Sorting these into ascending order is a trivial problem. We just scan along the line and notice which is the smallest, and put it on the left, and then repeat the process for the remaining digits.

The computer, however, needs a very precise set of instructions, so we shall have to think very clearly about what steps are required. Here's one approach: Compare the first digit with the second digit. If the first digit is bigger than the second one, swap them. If the first digit is smaller than the second one, leave their positions unchanged.

Compare the second digit with the third digit. If the second digit is smaller than the third one, leave their position unchanged.

Repeat the process of comparing pairs of digits until the last pair of digits has been compared.

If there were no swaps, all the numbers must be in order. If there were any swaps, go back to the beginning and repeat the process.

If you think about this process, you will see that it will indeed sort any group of numbers into ascending numeric order. Look at what would happen to our original set of numbers as each pair of digits is compared:

```
4  9  2  8  3
4  2  9  8  3
4  2  8  9  3
4  2  8  3  9
```

All the pairs have now been compared and swapped where necessary. Since at least one swap took place, go back to the beginning and repeat the process:

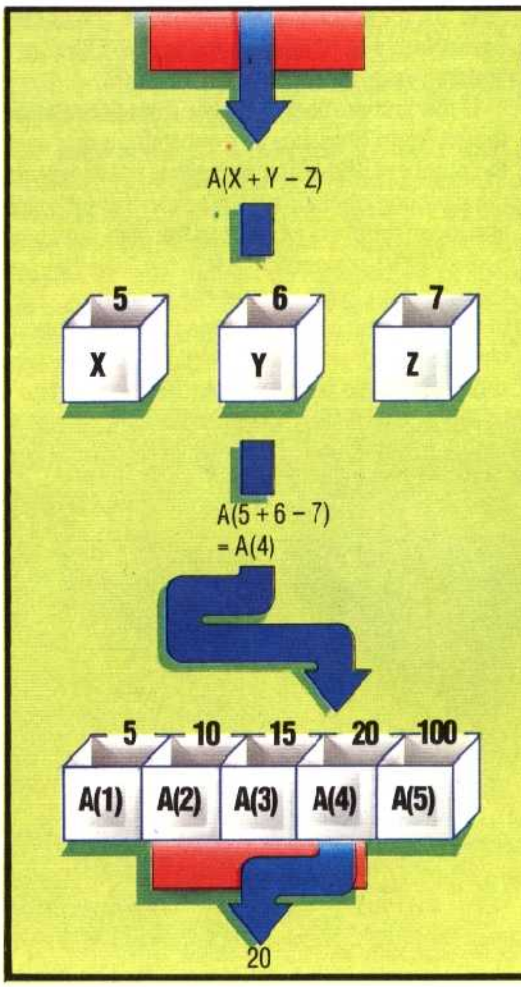
```
4  2  8  3  9
2  4  8  3  9
2  4  3  8  9
2  4  3  8  9
```

There were still swaps, so go back to the beginning and repeat:

```
2  4  3  8  9
2  3  4  8  9
2  3  4  8  9
```

There were no swaps, last time through, so every number must be smaller than the number to its right. The numbers must be in ascending order and the operation can be terminated.

Using subscripted variables allows a sort routine like this to be implemented easily in BASIC, because the subscript itself can be a variable. If our original five numbers were the values in an array; so that A(1) = 4, A(2) = 9, A(3) = 2, A(4) = 8 and



Subscripted Variables

Subscripted variables (variables with several 'compartments' in the box) increase the power of BASIC enormously. Here, variable A has the subscript X + Y – Z. Each of these is a variable, and the value of each is shown inside the small boxes. X has the value 5, Y is 6 and Z is 7. X + Y – Z is therefore equivalent to 5 + 6 – 7, which is 4. A(4) is the fourth element in the array. Its value is 20. PRINT A(X + Y – Z) will therefore result in 20 being printed on the screen

TONY LODGE