# Another Dimension

**One-dimensional arrays, as we have seen, store a collection of data that have something in common. Two-dimensional arrays are used for tables and charts**

So far we have considered two types of variables, simple variables and subscripted variables. Simple variables are like memory locations where numbers (or character strings) can be stored and manipulated by referring to the variable 'label'. Simple variables can store just one value or string and have 'simple' variable names — N, B2, X, Y3 are examples. Subscripted variables, sometimes called one-dimensional arrays, can store a whole list of values or strings. The number of values or strings that can be held is specified at the beginning of the program using the DIM statement. For example, DIM A(16) establishes that the array labelled A can contain 16 separate values. It should be noted, however, that many BASICS accept A(0) as the first element, so that DIM A(16) actually defines 17 elements. These 'locations' are referred to by using the appropriate subscript. PRINT A(1) will print the first element in the array; LET B = A(12) assigns the value in the 12th element in the array to variable B; LET A(3) = A(5) assigns the value of the fifth element to the third element.

Sometimes, however, we need to be able to manipulate data that is best presented as tables. Note how closely this resembles a spreadsheet (see page 158). Such data could range from tables of football results to a breakdown of sales by item and department in a store. As an example of a typical table of data, consider this breakdown of household expenditure over a one year period:

| | RENT | PHONE | ELECTR. | FOOD | CAR |
|---|---|---|---|---|---|
| JAN | 260.00 | 25.10 | 41.50 | 161.30 | 50.55 |
| FEB | 260.00 | 35.40 | 43.75 | 145.90 | 46.20 |
| MAR | 260.00 | 29.05 | 50.70 | 151.20 | 43.40 |
| APR | 260.00 | 26.20 | 44.60 | 155.30 | 49.20 |
| MAY | 260.00 | 19.30 | 39.80 | 150.95 | 48.30 |
| JUN | 260.00 | 20.45 | 32.60 | 147.65 | 52.30 |
| JUL | 260.00 | 30.50 | 26.10 | 150.35 | 58.40 |
| AUG | 260.00 | 29.50 | 22.40 | 148.05 | 61.20 |
| SEP | 260.00 | 28.25 | 24.45 | 148.60 | 59.45 |
| OCT | 260.00 | 31.15 | 34.50 | 154.90 | 23.50 |
| NOV | 260.00 | 31.05 | 39.50 | 160.05 | 45.95 |
| DEC | 260.00 | 28.95 | 42.20 | 210.60 | 51.25 |

Arranging the information in this way allows it to be manipulated in a number of ways relatively simply. It is easy, for example, to find the total expenditure in March by simply adding up all the figures in the row for March. It is just as easy to find the total expenditure for the year on the telephone or the car by adding up the vertical columns. Similarly, it is easy to find monthly or yearly averages. This table is called a two-dimensional array. It has 12 rows and five columns.

Two-dimensional arrays such as this can also be represented in BASIC in much the same way as single-dimension arrays. The difference is that the variable now needs two subscripts to reference any location.

If we were writing a BASIC program using this table of information, the simplest thing would be to treat the whole table as a single two-dimensional array. Just as with ordinary subscripted arrays, we give it a variable name. Let's call it A (for 'Array'). Again, as with ordinary subscripted arrays, it will need to be DIMensioned. As there are 12 rows and five columns, it is dimensioned thus: DIM A(12,5). The order in which the two subscripts are put is important; the convention is that rows are specified first and columns second. Our table above has 12 rows (one for each month) and five columns (one for each of the five categories of expenditure), it is therefore a 12-by-5 array.

The DIM statement serves two essential functions. It sets aside enough memory locations in the computer's memory for the array, and it allows each of the locations to be specified by the variable name followed, in brackets, by the row and column positions. The DIM statement DIM X(3,5), for example, would create a variable X able to represent an array with three rows and five columns.

Look at the table and assume that the information has been entered as the elements in a two-dimensional array labelled A. Find the values present in A(1,1), A(1,5), A(2,1), A(3,3) and A(12,3).

It is possible to enter a table of information as an array in part of a program by using LET statements, for example.

```
30 LET A(1,2) = 25.1
40 LET A(1,3) = 41.5
50 LET A(1,4) = 161.30
:
:
610 LET A(12,5) = 51.25
```

But this is clearly a laborious way of doing things. A far simpler method is to use either READ and DATA statements or the INPUT statement with nested FOR...NEXT loops. Let's see how it could be done using the READ statement:

```
10 DIM A(12,5)
20 FOR R = 1 TO 12
30 FOR C = 1 TO 5
40 READ A(R,C)
50 NEXT C
60 NEXT R
70 DATA 260, 25.1, 41.5, 161.3, 50.55, 260, 35.4,
```