```
   43.75
80 DATA 145.9, 46.2, 260, 29.05, 50.7, 151.2, 43.4,
   260
90 DATA 26.2, 44.6, 155.3, 49.2, 260, 19.3, 39.8,
   150.95
100 DATA 48.3, 260, 20.45, 32.6, 147.65, 52.3,
   260, 30.5
110 DATA 26.10, 150.35, 58.4, 260, 29.5, 22.4,
   148.05, 61.2, 260
120 DATA 28.25, 24.45, 148.6, 59.45, 260, 31.15,
   34.5
130 DATA 154.9, 23.5, 260, 31.05, 39.5, 160.05,
   45.95
140 DATA 260, 28.95, 42.2, 210.6, 51.25
150 END
```

There are a number of important points to note about this program. The first is that the DIM statement is right at the beginning of the program. A DIM statement should be executed only once in a program and so it is usual to place it near the beginning or before any loops are executed. The second point to note is that there are two FOR...NEXT loops, one to set the 'row' part of the subscript and one to set the 'column'. These two loops do not follow one after the other; they are 'nested' one inside the other. Notice the limits chosen. FOR R = 1 TO 12 will increment the value for the row from one to 12; FOR C = 1 TO 5 will increment the value for the column from one to five.

Right in the middle of the nested loop is the READ statement. The crucial part of the program is:

```
20 FOR R = 1 TO 12
30 FOR C = 1 TO 5
40 READ A(R,C)
50 NEXT C
60 NEXT R
```

The first time through, after lines 20 and 30 have been executed, the values of R and C will both be one, so line 40 will be equivalent to READ A(1,1). The first item of data in the DATA statement is 260, so this value will be assigned to the first row and the first column of the array. The choice of eight elements to each DATA statement is purely arbitrary.

After that has happened, the NEXT C statement sends the program back to line 30 and the value of C is incremented to two. Line 40 is now equivalent to READ A(1,2) and the next item of data, 25.1, will be assigned to the first row and the second column of the array. This process is repeated until C has been incremented to 5. After that, the NEXT R statement in line 60 returns the program to line 20 and R is incremented to two. Line 30 will set C to one again and so now line 40 will be equivalent to READ A(2,1).

Nesting loops in this way is very useful, but care is needed. Each loop must be nested completely within another loop and the order of the NEXT statements must be carefully observed. Notice how the first loop, FOR R, has the second NEXT statement. When there are two loops, one nested inside the other, the first loop is called the outer loop and the second is called the inner loop. The whole of the inner loop will always be completed before the index of the outer loop is incremented. It is possible to nest loops to as many 'depths' as required by the program, but such programs can become complex and difficult to follow and debug. It is bad programming practice to put branching instructions inside loops and GOTOs are to be avoided.

Let's look at the DATA statements. Notice that commas are used to separate data items, but there must be no comma before the first data item or after the last. We have inserted spaces between each data item, but this is not normal. Mistakes when entering the data are easy to make and difficult to spot later. As many DATA statements as required may be used. Each new line needs to start with a DATA statement. The data is read in one item at a time, starting from the beginning of the first DATA statement and working through until all the items have been read. Be sure that the number of data items is correct or you will get an error message when the program is run.

The program presented so far does not actually do anything except convert appropriate data into a two-dimensional array. After the program has been entered and RUN, nothing will apparently happen and all you will see on the screen will be the BASIC prompt. To test that the data is correctly placed, try a few PRINT commands. (A command in BASIC is a keyword that can be immediately executed without having to be within a program and does not therefore need a line number. Examples are LIST, RUN, SAVE, AUTO, EDIT and PRINT). PRINT A(1,1) <CR> should cause the number 260 to appear on the screen. What will be printed by the following commands?

```
PRINT A(12,1)
PRINT A(1,5)
PRINT A(5,1)
PRINT A(5,5)
```

To make the program do something useful, it will need to be extended. As it stands it forms an adequate basis for a 'main program'. To use it as part of a larger, more useful program, modules can be written as subroutines to be called by GOSUBs inserted at suitable points before the END statement.

In the early stages of designing a household accounts program, it is best to start with a simple written description of the general requirements. We might decide that we want to be able to have totals and averages calculated for monthly expenditure or by category (electricity, for example). We can work out the details of how to derive these results at a later stage. If there is a choice to be made within the program about which subroutines we wish to be executed we will probably want to be prompted by a 'menu' which will direct control to the appropriate subroutines as a result of our response. An early sketch of the program at this stage might look like this: