the fields. But here's a way to 'cheat' if you want to use multi-dimensional arrays and your BASIC cannot handle them.

The trick is to treat all the elements that would be in the multi-dimensional array as though they were elements in a one-dimensional array. For example, a two-dimensional array with three rows and five columns would be dimensioned like this: DIM A(3,5) and would contain a total of 15 elements: A(1,1) to A(3,5). The same information could be held in an ordinary subscripted array like this: DIM A(15). Everywhere that a two-dimensional array referred to A(R,C), we would substitute A((R-1)*5+C).

If we use a separate string array for each field, we have to decide how to DIMension the arrays. The simplest way is to use a fixed array size, but this limits the total number of records we can store in the database. A better approach would be to set the array size according to how many records there are in use. Not all dialects of BASIC, however, allow string arrays to be as big as you would like. Even if they did, a large number of records in the database could soon use up all the available memory in the computer. Here is a program that will enable you to find out the maximum number of elements your computer will allow. Many versions of BASIC, however, will allow as many elements in an array as you want, right up to the point where all available memory is used up. Each time the program asks you 'WHAT ARRAY SIZE?' enter a larger value until you eventually get an error message. The CLEAR in line 100 has the effect of eliminating the array at the end of each pass. Without this statement, you would get an error message in line 30 through attempting to re-dimension an array.

```
10 READ D$
20 INPUT "WHAT ARRAY SIZE";A
30 DIM N$(A)
40 FOR L = 1 TO A
50 LET N$(L) = D$
60 NEXT L
70 FOR L = 1 TO A
80 PRINT L, N$(L)
90 NEXT L
100 CLEAR
110 GOTO 10
120 DATA "HOME COMPUTER COURSE"
130 END
```

Even if only 40 characters were allowed in each element, with five fields per record, and if there were 256 elements set aside for each array, the amount of memory required to hold all the data within the main memory becomes prodigious. If one byte is required for each character to be stored, we would need 51,200 bytes (5×40×256 bytes) for the data alone. It is obviously not practical to use up so much main memory on the data, and that's why separate data files are used.

Unfortunately, as we have already suggested, file handling routines can be a little difficult to use. If we wish to avoid using external files, the only alternative is to put the data in a DATA statement so that it is always present in the program. Apart from the strain this imposes on the computer's memory capacity, this technique makes modification to the data extremely tedious and prone to error. Therefore it is preferable to use external data files. Once you have tried a few short programs to write data to and from external files, however, the whole process will become much clearer and easier to understand. By way of illustration, we have chosen two very different machines and versions of BASIC to supplement our short daily temperature program given in Microsoft BASIC. These are for the Sinclair Spectrum and the BBC Micro. Both these versions of BASIC differ considerably from our usual Microsoft BASIC, and readers are referred back to the 'Basic Flavours' boxes in earlier parts of the Basic Programming course for details on some of these differences.

In Spectrum BASIC, OPEN# and CLOSE# are reserved for use with the Microdrive. When cassette storage is used, special versions of the SAVE and LOAD commands are needed. The ordinary SAVE command is used for storing programs and program variables on tape (and, of course, ordinary data in DATA statements). Arrays can be saved on tape using the SAVE-DATA statement. This takes the form:

```
SAVE filename DATA array name()
```

The filename is the name given to the file (TEMP.DAT in the Microsoft program). The array name is simply the name of the string array followed by a pair of closed and empty brackets. To SAVE the daily temperature results, we would first have to create a DIMensioned string array and write the data into it, perhaps using READ-DATA statements. To make the difference between the filename and array name more apparent, we will call the array c$ and the filename will be "TEMPDAT".

```
10 DIM c$(14,4)
20 FOR x=1 TO 14
30 READ c$(x)
40 NEXT x
50 DATA 1,13.6,2,9.6,3,11.4,4,10.6,5,11.5,6,11.1,7,10.9
60 SAVE "TEMPDAT" DATA c$()
70 STOP
80 LOAD "TEMPDAT" DATA c$()
90 FOR L=1 to 14 STEP 2
100 PRINT "DAY":c$(L),c$(L+1)
110 NEXT L
120 STOP
```

Line 60 saves all the data in the string array c$ in a data file with the filename TEMPDAT. The program will then stop at line 70, and you should rewind the tape. The keyword CONT will restart the program. Line 80 reverses the process and stores the data in TEMPDAT in the c$ array.

The BBC Micro has one of the most sophisticated dialects of BASIC available on home computers. It allows structured programming with such advanced features as a REPEAT-UNTIL