New Entries

In order to insert a new entry in an array, it is first necessary to find a blank space. The binary search is an efficient way of achieving this

We saw in the previous instalment how a file of data is made up of records, which are divided into fields, each of which can be given access to the other fields through an indexing field. Now we will consider some of the techniques of searching through these lists.

Creating records for our address book is not difficult. Assume that a separate string array exists for each of the fields in the record. These can be called FLNAME$ (for full name), STREET$, TOWN$ and PHONE$ (we'll talk later about our use here of a string variable rather than a numeric variable for the phone number field). In the list of eight desirable functions of the address book program, number six was the facility to add new entries. If these eight choices were presented on the screen at the beginning when the program was run, selecting 6 would take you to an input routine of the type presented as an exercise.

Assume there are already a number of entries in the address book, but you can't remember how many. It is essential that new entries are not written over existing entries, so one of the tasks of the program might be to search through the elements in one of the arrays to find the first one containing no data.

Searching through an array to see whether an element is 'occupied' is not difficult. String variables can be compared in BASIC just as numeric variables can. IF A$="HOME" THEN ... is just as valid as IF A=61 THEN..., at least in most versions of BASIC. If any of the arrays in our address book has an entry already, this will consist of at least one alphanumeric character. An 'empty' element will contain no alphanumeric characters, so all we need to do is search through the elements, starting at the beginning, until we find one containing no characters.

If there are arrays for the name, the street, the town and the phone number, we will have four arrays with one element in each for each field in the record. Since all these fields 'go together', the 15th record will have its name data in the 15th element of the name array, its street data in the 15th element of the street array, the town data in the 15th element of the town array, and the phone number data in the 15th element of the phone number array. We therefore need only to search through one of these arrays to find an empty element; we don't need to check all the arrays.

If the variable POSITION represents the number of the first free element in any one of the arrays, a program to locate POSITION (assuming it is not already known) could be as simple as this:

```
PROCEDURE (find free element)
BEGIN
  LOOP
    REPEAT UNTIL free element is located
    READ Array (POSITION)
    POSITION = POSITION + 1
    IF Array(POSITION) = " "
      THEN note POSITION
      ELSE do nothing
    ENDIF
  ENDLOOP
END
```

In BASIC, this could be as simple as:

```
1000 FOR L = 0 TO 1 STEP 0
1010 LET POSITION = POSITION + 1
1020 IF FLNAME$(POSITION) = " " THEN LET
  L = X
1030 NEXT L
1040 REM rest of program
```

Note that the value of X in line 1020 is the value required to terminate the FOR...NEXT loop and this value varies from machine to machine (see Basic Flavours). It is also important to note that this is a program fragment, and it is assumed that FLNAME$() is DIMensioned and that POSITION has been initialised. To run this fragment as a program on its own, you must DIMension FLNAME$() and initialise POSITION and X, at some point before line 1000.

Although we have used the FOR X = 0 TO 1 STEP 0 technique before, this is a good place to examine in more detail how it works. Usually, a FOR...NEXT loop in BASIC 'knows' beforehand how many times the program fragment is expected to repeat. If you want to repeat something 30 times, FOR X = 1 TO 30 will do admirably. This time, however, we are simulating a REPEAT...UNTIL loop. Although ordinary versions of BASIC do not have REPEAT...UNTIL on offer, it is easy enough to simulate using FOR...NEXT. As long as the test in line 1020 fails, L (the FOR...NEXT loop counter) remains at the value 0, with 0 added to it at every iteration (repetition of the loop); while line 1010 causes POSITION to be increased by 1 every iteration. When the test in line 1020 is true (that is, when an empty element of FLNAME$() is found), L is set to the value X, and the FOR...NEXT loop is terminated at line 1030. This leaves POSITION pointing to the first free element of FLNAME$().

POSITION is a value we are likely to need to