programs will start by calling FINDREC. This sub-program is based on a search routine similar to the one described on page 273. The chief difference this time is that (in all probability) no two data items will be identical, since few people have completely identical names.

There are two ways a search can be conducted. One is to search through an unordered pile. This makes the searches slower than they need to be. In the worst case, the routine might have to search through all of the data items before locating the item being searched for. Searching through an unordered pile does have the advantage, however, that sort routines are not required every time a record is added, deleted or modified.

If the data is ordered in some way — either numerically or alphabetically, for example — the program will have to search through only a small fraction of the items in the list. The longer the list is, the more efficient a binary search becomes compared with searching through an un-ordered pile. In fact, if there is enough data in the file to warrant it, the sorting of the records after a modification can be speeded up by conducting a preliminary search to locate the first and last occurrence in the array of the initial letter of the surname in the record involved.

Another way to speed up the sort routine might be to maintain a look-up table of the locations in the array of the first occurrence of each letter of the alphabet. This table, however, would need to be carefully maintained (updated) whenever any changes were made to the data.

The subject of searching and sorting is one of the largest areas in programming, and books have been devoted to it. We will not attempt to find the optimal solution for our address book program since this depends on a large number of factors, including the number of records in the file and whether or not disk drives are available.

A program in pseudo-language for a search through the elements in the MODFLD$ array is now given. The string variable KEY$ is the key for the search. The term 'key' here means the identifying group of characters used to specify which record (or records) is required.

```
Prompt for name to be searched
LET KEY$ = name (to be searched)
LET BTM = 1
LET SEARCHING = 0
LET TOP = SIZE
LOOP while (BTM < = TOP) AND (SEARCHING = 0)
   LET MID = INT ((BTM + TOP)/2)
   IF KEY$ = MODFLD$(MID)
      THEN
         PRINT NAMFLD$(MID)
         PRINT STRFLD$(MID)
         PRINT TWNFLD$(MID)
         PRINT CNTFLD$(MID)
         PRINT TELFLD$(MID)
         LET SEARCHING = 1
      ELSE
         IF KEY$ > MODFLD$(MID)
            THEN LET BTM = MID+1
            ELSE LET TOP = MID–1
         ENDIF
   ENDIF
ENDLOOP
IF SEARCHING = 0 THEN PRINT "RECORD NOT
   FOUND"
END
```

This piece of pseudo-language is closely based on the program used for searching football scores on page 275, but you will see that it does have a suitable output if the record cannot be found (the last PRINT statement), which will be executed only if the loop fails to locate an exact match between KEY$ and MODFLD$(MID).

Unfortunately, an exact match is rather unlikely, even if the name and telephone number you want is in the database. This is because the IF KEY$ = MODFLD$ statement is totally inflexible; it does not allow for the slightest difference between the character string input by the user in response to the prompt and the character string stored in MODFLD$(MID). In an ordinary address book, the eye scans down the page and is able to allow for all sorts of small differences in the actual representation of the record and what you are looking for. The computer cannot do this.

There are, however, ways of avoiding this, although they all involve extra programming effort and will take a little more time to run. The first improvement would be to check only the surname first, and for this reason it makes sense for the name stored in MODFLD$ to be in the form SURNAME (space) FORENAME. We developed a routine for reversing the order of a name earlier in the Basic Programming course (see 'Basic Flavours') and this can be incorporated as a subroutine within the ADDREC routine when the MODFLD$ field is created.

Having successfully located the first occurrence of the required surname, the FINDREC routine should then check the forename part of that element to see if it is identical to the name input (KEY$). If it is, there is no problem — the record has been located. If it is not, however, the problem starts to get complicated, and we have to plan our strategy carefully. We could, for example, search through all the forenames, and if an exact match is not found, start looking for an approximate match. The difficulty here is: what exactly constitutes an approximate match?

Instead of the "RECORD NOT FOUND" message in the program above, it might be better to give a message like "EXACT MATCH NOT FOUND, TRY FOR A CLOSE MATCH? (Y/N)?" What do the words 'close match' mean? Is Bobby a close match to Robert? How about Robrt? Both of these represent possible inputs in the FINDREC program. Let's try to define what we mean by a close match and then start to develop a program in BASIC to find the closest match to an input string.

Suppose the string in memory was ROBERT. Which of the following is the closer match: ROB or RBRT? The second gets four letters right out of six,