should always be done for local variables to prevent 'side effects' from the use of the same variable elsewhere in the program.

When the program is first run, various types of initialisation will take place and data will be loaded from disk or tape and transferred to string variables. The CHOOSE menu will then be presented. If the user chooses option 6 (to add a record to the file), the value of variable CHOICE returned will be 6, and this will call the sub-program ADDREC. ADDREC will assume that SIZE has already had a value assigned to it and so it can start prompting for inputs (note: this also assumes that INITIALISE has already correctly DIMensioned the necessary arrays).

Adding a new record also means that the file is now, potentially at least, out of order. Since a sort may take some time, it may not be necessary to sort the records after each addition has been made – that is a decision we shall defer for the moment. Instead, we will set a flag to indicate that a new record has been added.

We are now in a position to start making a tentative list of possible arrays, variables and flags that may be needed by the program.

ARRAYS

NAMFLD$	(name field)
MODFLD$	(modified name field)
TWNFLD$	(town field)
CNTFLD$	(county field)
TELFLD$	(telephone number field)
NDXFLD$	(index field)

VARIABLES

SIZE	(current size of file)
CURR	(index of current record)

FLAGS

RADD	(new record added)
SORT	(sorted since record modification)
SAVE	(save executed since record modification)
RMOD	(modification made since last save)

It is likely that in the course of developing the program a few more arrays will be needed. Certainly more variables will be needed. As for the flags, it is apparent that although others will be necessary, the four given above may not all be required. There will be no need either to save or sort the file (assuming it is already saved and sorted) unless a modification has taken place, so RMOD is possibly the only flag really needed. But if we do decide to use all four flags, the INITIALISATION sub-program should set them all to their appropriate values. As further practice in top-down program refinement, let's see how easy it is to code *ADDREC*.

I 4 (EXECUTE) 6 (ADDREC)

```
BEGIN
   Locate current size of file
   Prompt for inputs
   Assign inputs to ends of arrays
   Set RMOD flag
END
```

II 4 (EXECUTE) 6 (ADDREC)

```
BEGIN
   (size of file is SIZE)
   (prompt for inputs)
   Clear screen
   Print prompt message for first array(SIZE)
   Input data to array(SIZE)
   (prompt and input for all arrays)
   Set RMOD to 1
END
```

All this is straightforward and does not involve loops or other complicated structures. The next step can be direct coding into BASIC. The only points to note are that SIZE is a variable set during the execution of INITIALISE and does not need to be coded as part of this section.

III 4 (EXECUTE) 6 (ADDREC) BASIC CODE

```
CLS: REM OR USE PRINT CHR$(24) ETC TO CLEAR
   SCREEN
INPUT "ENTER NAME";NAMFLD$(SIZE)
INPUT "ENTER STREET";STRFLD$(SIZE)
INPUT "ENTER TOWN";TWNFLD$(SIZE)
INPUT "ENTER COUNTY";CNTFLD$(SIZE)
INPUT "ENTER TELEPHONE NUMBER";
   TELFLD$(SIZE)
LET RMOD=1
LET NDXFLD$=STR$(SIZE)
GOSUB *MODNAME*
RETURN
```

The third to last line sets the NDXFLD$ field to the value of SIZE (converted into a string by STR$), so that it can act as an index at a later stage. The subroutine *MODNAME*, called just before the end of the program, is none other than the program described in detail on page 254. A few slight changes will be needed to that program, but these are just details. This subroutine has the function of taking the ordinary (fuzzy) name input and converting it into a standard form. The output from this subroutine will be an element (SIZE) in an array called MODFLD$. All name searches and sorts can now be conducted on the elements in MODFLD$, and since the element will have the same index as the other fields in the record, it will be easy to display the name and address as they were originally entered. In other words, the search will be made on MODFLD$ but the display will come from NAMFLD$.

That's about all that's involved in adding a new record to the file, although we have not made allowances for any error checking, or provision for what would happen if there is no more space left in the array. Since all our programs are being written in modular form, modifications and improvements such as these can easily be made later without having to rewrite the whole program.

The sub-programs MODREC and DELREC (to modify and delete records respectively) are fairly similar to ADDREC, except that before they can be executed we have to locate the record we want to change. Consequently, both of these sub-