same way conceptually — one file containing many records, each record containing many fields — just like our computerised address book.

A sequential file on a disk or cassette tape, however, doesn't care how the information in it is used or organised by the program. Data files just contain a series of data items, and each individual item of data is called a record. A single record in a data file wouldn't, therefore, normally correspond to a record in the database sense of the word.

It's up to the program to read in records from the data file and assign them to variables or arrays. These variables and arrays need to be organised to form a 'conceptual' record containing a limited set of related information. There is no one-to-one relationship between the records in a data file and the records comprising a database.

Once the set-up program has been run it should never be needed again. In fact, if it ever were run again it would destroy any 'legitimate' data you might have entered in the address book database. We will see why this would happen when we look at the modified *RDINFL* program.

When the program is run it does not 'know' if there is legitimate data in the data file or not. The first thing *RDINFL* does is to OPEN the 'ADBK.DAT' file and read in the first record (or data item). This is not read into an element in an array, as you might expect, but into a special string variable we have called TEST$. Before any other records are read in, TEST$ is checked to see if it contains the string @FIRST. If TEST$ does contain @FIRST, the program knows there is no valid data in the file and so there is no point in trying to read in any more data and assign it to arrays. Consequently, the file can be closed and the rest of the program can continue. Since there is no valid data in the file, the user can do nothing useful until at least one record has been entered and so the value of TEST$ can also be used to force the program to go to the *ADDREC* subroutine so that at least one valid record will be added before anything else can be done.

If, on the other hand, the value of TEST$ is not @FIRST, the program can assume that there is valid data in the file and can start assigning the data to the appropriate arrays. The modified *RDINFL* subroutine follows:

```
1400 REM *RDINFL* SUBROUTINE
1410 OPEN "I",#1,"ADBK.DAT"
1420 INPUT #1,TEST$
1430 IF TEST$ = "@FIRST" THEN GOTO 1530: REM
     CLOSE AND RETURN
1440 LET NAMFLD$(1) = TEST$
1450 INPUT #1,MODFLD$(1),STRFLD$(1),TWNFLD$
     (1),CNTFLD$(1),TELFLD$(1)
1460 INPUT #1,NDXFLD$(1)
1470 LET SIZE = 2
1480 FOR L = 2 TO 50
1490 INPUT #1,NAMFLD$(L),MODFLD$(L),STRFLD$
     (L),TWNFLD$(L),CNTFLD$(L)
1500 INPUT #1,TELFLD$(L),NDXFLD$(L)
1510 REM  SPACE FOR CALL TO 'SIZE'
     SUBROUTINE
1520 NEXT L
1530 CLOSE #1
1540 RETURN
```

Line 1420 assigns a single record from the ADBK.DAT file to the variable TEST$. The next line then checks this to see if its value is @FIRST. If it is, a GOTO is used to jump to the line that closes the file (line 1530) and then the subroutine RETURNs to the calling program. No further attempts are made to read in data. Assuming that there is no valid data in the file, program control will be returned to *INITIL*, which then calls *SETFLG*. All this routine does at the moment is to set the value of SIZE to 1 if TEST$ = @FIRST. The code for *SETFLG* is given below. Note that there are several REMs to allow space for further flag setting should we want to do this later.

```
1600 REM  *SETFLG*
1610 REM  SETS FLAGS AFTER *RDINFL*
1620 REM
1630 REM
1640 IF TEST$ = "@FIRST" THEN LET SIZE = 0
1650 REM
1660 REM
1670 REM
1680 REM
1690 RETURN
```

SETFLG then RETURNs to *INITIL*, which in turn RETURNs to the main program. *MAINPG* then calls *GREETS*, which displays the greeting message. *GREETS* does not need any modification from the previously published version of it.

The next routine called by the main program is *CHOOSE*. A very small modification to the *CHOOSE* subroutine on page 357 will establish a way of forcing the user to add a record if the program is being run for the first time.

```
3500 REM  *CHOOSE* SUBROUTINE
3510 REM
3520 IF TEST$ = "@FIRST" THEN GOSUB 3860
3530 IF TEST$ = "@FIRST" THEN RETURN
3540 REM  'CHMENU'
3550 PRINT CHR$(12)
3560 PRINT "SELECT ONE OF THE FOLLOWING
3570 PRINT
3580 PRINT
3590 PRINT
3600 PRINT "1. FIND RECORD (FROM NAME)"
3610 PRINT "2. FIND NAMES (FROM INCOMPLETE
     NAME)"
3620 PRINT "3. FIND RECORDS (FROM TOWN)"
3630 PRINT "4. FIND RECORD (FROM INITIAL)"
3640 PRINT "5. LIST ALL RECORDS"
3650 PRINT "6. ADD NEW RECORD"
3660 PRINT "7. CHANGE RECORD"
3670 PRINT "8. DELETE RECORD"
3680 PRINT "9. EXIT & SAVE"
3690 PRINT
3700 PRINT
3710 REM  'INCHOI'
3720 REM
3730 LET L = 0
3740 LET I = 0
3750 FOR L = 0 TO 1
3760 PRINT "ENTER CHOICE (1 - 9)"
3770 FOR I = 1 TO 1
3780 LET A$ = INKEY$
3790 IF A$ = "" THEN I = 0
3800 NEXT I
3810 LET CHOI = VAL(A$)
3820 IF CHOI <1 THEN L = 0 ELSE L = 1
3830 IF CHOI >9 THEN L = 0
3840 NEXT L
3850 RETURN
```

Two lines have been added. The first tests TEST$. This variable still contains the value read into it in the *RDINFL* routine. If it is @FIRST we know that there is no valid data in the file and so the only appropriate option is ADDREC, which is number 6. If the test is passed, control is passed to *FIRSTM*, a routine that displays an appropriate message and sets the CHOI variable to 6. When the subroutine returns to line 3530, TEST$ is tested again (it is