Dummy Run

In order to use data files it is first necessary to create them in skeleton form, and then fill them with information

At the end of the last instalment of the course, readers were left with the problem of solving this apparent dilemma: how can we make a program read in a file that does not exist (on tape or disk) when the program is first run? The initial activity we are likely to want the program to perform will be to read in the data file and assign this data to arrays or variables. Yet, if we insist on writing to the file first, whenever the program is run, we will have to be very careful in the programming not to lose all the data in the file. As we discovered last time, attempting to open a non-existent file will either simply not work, or else cause the program to 'crash' (stop functioning).

Fortunately, there's a very simple solution. Many commercial software packages include an 'install' or 'set-up' program that has to be run before the program proper can be used, and this is the approach that we shall adopt. Such programs typically allow the user to do a small amount of 'customising' (such as selecting whether the printer to be used will be an Epson or a Brother, parallel or serial, and so on), but they also create data files that will later be used by the main program. Remember, unlike program files, data files can be accessed by any program (see page 316).

To solve our problem and allow '*RDINFL*' (the routine that reads in the file and assigns the data to the arrays) to be performed, we can write a very simple set-up program that does nothing more than open a file and write a dummy value into it. We will choose a value that can be subsequently recognised by the program proper as not being a valid address book record. A suitable value would be the character string @FIRST, because no name or address, no matter how obscure its origin, is likely to start with this particular string. *RDINFL* will have to be slightly modified so that when it opens and reads in from the file, it tests for this value before going any further. If your computer doesn't have the @ symbol, then you will have to replace it with '!' or another character — as long as this is a string that won't occur naturally in your address book. First, however, here is the set-up program:

```
10 REM THIS PROGRAM CREATES A DATA FILE
20 REM FOR USE BY THE ADDRESS BOOK
   PROGRAM
30 REM IT WRITES A DUMMY RECORD THAT CAN
40 REM BE USED BY *RDINFL*
50 REM
60 REM
70 OPEN "O",#1,"ADBK.DAT"
80 PRINT #1,"@FIRST"
90 CLOSE #1
100 END
```

As mentioned previously in the Basic Programming course, the details of reading and writing files differ considerably from one version of BASIC to another, but the principle is almost always the same. First, the file must be declared OPEN before it can be used for either input or output. Then the direction of data flow is declared, either IN or OUT. Next a 'channel' number is assigned to the file. This allows more than one file to be open and in use at the same time (for the time being, however, we will use only one file). Finally, the name of the file we wish to use must be declared.

Line 70 in the program (left) is in Microsoft BASIC and is similar in principle to the OPEN statements used by most BASICS (BBC BASIC is somewhat different — see page 319). OPEN, of course, declares that a file is to be OPENed and 'O' says that data will be output. #1 is the number we are assigning to the file for this operation; a different file number could be used later if needed. 'ADBK.DAT' is the name we have given to the file.

Line 80 simply writes a single record to the file. The syntax of writing data to a file is usually (in most BASICS) exactly the same as the syntax used for PRINTing, except that the PRINT statement must be followed by the file number — #1 in this case.

Line 90 CLOSEs the file. Files may be left open for as long as needed in the program, but 'open' files are very vulnerable and should be CLOSEd as soon as possible within the program in order to protect the data in them. If, for example, you were to accidentally switch off the computer while the file was open, you could find that data has been lost when you next read the file.

There is some confusion over the way the terms record and file are used in computers, and this confusion is worst when we are talking about databases, on the one hand, and data files on the other. In a database, the file is a whole set of related information. Using the analogy of an office filing cabinet, the file could be a drawer labelled PERSONNEL. This file could comprise one record (a card in a folder) on each person in the company. Each record (card) would contain a number of fields, identical for each record, containing such information as NAME, SEX, AGE, SALARY, YEARS OF SERVICE etc.

If the PERSONNEL file were computerised, all the information would be treated in exactly the