FOR THE RECORD

In this series on file handling we've looked at the basic principles behind all computer filing systems. However, it is important to realise that file handling methods tend to be machine-specific. In this final instalment, therefore, we examine the ways in which these techniques may be used on cassette-based micros.

Different micros use different file handling techniques, and therefore it is often necessary to adapt a standard program to run on a particular machine. For this reason, it is important to understand how the facilities and commands offered by your own system relate to the general methods we have discussed. As an example, let us examine the storage of data files on a standard cassette-based micro. The first point to notice is that cassette systems, by their very nature, cannot handle random access files (see page 244), so data must be accessed in the order in which it is stored — that is, sequentially.

As sequential file handling involves reading information from one file, working on this information and then writing the modified data into a second file, it is obvious that two files must be in use ('open') at the same time. A cassette recorder is incapable of moving directly and accurately between two tape positions, so this 'two-file' system is not suitable for most cassette-based micros. The exceptions to this rule are those few micros, such as the Newbrain and Commodore PET range, that provide two cassette ports — one for reading, one for writing.

Most home machines, therefore, are limited to one sequential file at a time. In practice, this imposes some major limitations. The file must be read into memory before it is used, and then, if changes have to made, it must be written out onto cassette again. This may be done at intervals during program operation, or once at the end of the program run. Data files must be small enough to reside in RAM after space has been taken up by the filing program itself. Most home micros are thus restricted to small data files.

Three main methods have evolved for storing information on cassette. The simplest system does not use separate data files at all; instead all current variables are stored along with the program whenever the SAVE command is used. This method is used on the ZX81 and is also available on the Sinclair Spectrum. When a new data file is required, a 'fresh' copy of the program is used and then SAVEd along with its data. When this version is next LOADed, the data is automatically read back into the required variables. The virtue of this method is its simplicity — all the user has to do is to make sure that the complete program is correctly SAVEd and LOADed.

A slightly more sophisticated system requires a BASIC that is able to store and read back specific arrays. On the Oric Atmos, for example, the command STORE A$, "NAME" will write the array A$ to tape, and RECALL A$, "NAME" will read it back again. The whole array (A$(1), A$(2), etc.) is SAVEd, although the STORE and RECALL commands don't actually specify the array size, which is given automatically when the array is DIMensioned at the start of the program.

One problem with this system is keeping track of the number of entries in the array that are used in your program. One solution is to store the record count in the array before it is SAVEd. Most machines allow a zero subscript, so an element such as A$(0,0) can be used for the record count. The record count will be a numeric variable (in our program we use R), but if a string array is being used this must be converted into a string. This is done quite simply with a line such as: A$(0,0) = STR$(R). Once the array has been reloaded into the computer, R is reset with: R = VAL(A$(0,0)).

Although many micros do not support sophisticated file handling procedures, these may be simulated, as the listings here show. Once the file is safely installed in memory, it is easy to use BASIC arrays to treat it as a random access file.

Let's assume that a two-dimensional string array is used to store the data; this may be set up with a command such as DIM A$(100,3). The first subscript in the array may be used to refer to a particular record, and the second subscript will point to one of four fields. This allows the data to be stored in the familiar table format and is equivalent in operation to a random access file.

Another useful facility present on some machines is the APPEND command. This allows you to add data to the end of the sequential file without first reading through it and then creating a new version. Some computers have a command that allows a number of fields in a sequential file to be skipped over and this provides a simple random access facility.

This series has concentrated on the fundamental aspects of a complex topic. File handling is very much machine-dependent, and all the ideas that we have presented in these articles will need to be adapted for your own machine. But the basic principles will be the same, no matter what micro you use, and you should find much of the material useful when writing your own programs.