CONTINUING SERIAL

In the last instalment, we introduced the concept of sequential (or serial) files. We looked in detail at how they are constructed and discussed how they are manipulated by the operating system. Here we look at methods for using serial files in your own programs, and show how to overcome some of their inherent limitations.

A sequential file is a solid block of data on a disk or tape, and as such there are limitations as to how the file can be accessed and updated. To retrieve any one item, you must first read through all the preceding data. To update the file, it is usually necessary to make a copy of the file up to the point where changes are needed, then append the alterations to the new file and resume copying the original file immediately after the changes.

It is important to realise that information has to be organised in a suitable way inside the file. The choice of how this is done is up to the programmer and will depend on the application at hand. If it is a file containing English text then it is likely that it will be just a sequence of ASCII codes followed by an end of file marker. However, if the file is to contain a database such as a catalogue of books, then the information needs to be organised appropriately. The common way to do this is to divide the file into *records* and *fields*. Each book has its own record (entry) in the file and within each record are a number of fields, such as the book's title, author, publisher and so on. Within a sequential file, these divisions have to be marked by using special characters placed between items of data.

This is usually done by using a carriage return character (ASCII code number 13) to act as a marker between the fields and records. Since the file will have the same number of fields in each record, it's easy for the program to keep track of where a record ends and a new one begins.

Once a sequential file has been created, you need to be able to access and update it. The basic operations in filing are: retrieving records, adding records, deleting records and amending (editing) records. The diagrams show the various ways of achieving these with sequential files. Because you can only read through a sequential file in order and can't freely change data within it, these operations work by reading through the file, creating a new copy as they go. Any information that is to be changed is then written at the appropriate times into the new file as it is created. Finally, the new file becomes the current file and the old one is either discarded or kept on as a 'back-up' copy.

These simple techniques are the basis of all sequential filing routines. However, they make a major assumption about the capabilities of the operating system — that it can have two different files open at once in order to read from one and write to another simultaneously. This is not possible on all disk systems and is possible only on those cassette-based micros that have two cassette recorders attached. Both the Grundy Newbrain and Commodore PET range feature twin cassette interfaces for exactly this reason. Machines with single cassette drives are limited to files small enough to be read into memory in their entirety and processed there.

These methods of file handling also have an interesting side effect. After any alterations have been made to the file (either additions, deletions or amendments) you have two copies of the file: an old one that was the file before it was updated and a new copy with the changes. It is standard business practice to retain both files so that if something should happen to the new one, there is still a copy that is only one set of changes (or generation) out of date. In fact, most businesses keep three generations of any given file: the new file is called the 'son' file and the preceding file is kept on as the 'father' file. The file that was used as the basis for the father file is known as the 'grandfather' file.

These techniques can work with files that are too big to fit into the computer's memory in their entirety, because only a portion of the file, usually a handful of records, is actually being processed at any one time. With small files, however, much better performance can be gained by reading the whole file into arrays in memory and processing it there. All the file operations can be carried out at high speed in memory before the complete new file is written back to disk or tape.

This approach has one major danger — changes to the file are made permanent only when the information is written back to cassette or disk and, therefore, data could be lost if the program or computer is crashed or switched off while running. If you are using programs that work in this way, you should make sure that you frequently write copies of the file to storage and that a current copy has been made before the program is terminated.

A little experience with sequential file handling will show you that the techniques involved, although cumbersome, are mostly common sense. On many small systems, sequential files are the only file structure provided. When we move on to look at random access files, we'll discover techniques that complement serial files by providing simple and fast access and updating.