Search Warrant

The time taken to locate a particular record can be greatly reduced using the 'binary search' — provided that the file has already been sorted into an appropriate order

The three most important activities in the address book program — adding new records, saving the file on tape or disk, and reading in the file from mass storage when the program is first run — have now been developed. But an address book is no use if you can only add information and cannot extract any. What is needed next is a routine to find a record.

Finding a complete record from a name is likely to be the most frequent activity, and that's why the first option on the choice menu (*CHOOSE*) is FIND RECORD (FROM NAME). Searching is a highly important activity in many computer programs, especially in database programs where specific items of data often need to be retrieved from a file. Broadly speaking, there are two search methods — linear and binary. A linear search looks at each element in an array, starting at the beginning, and carries on until the particular item is located. If the data items in the array are in an unsorted state, a linear search is the only type that can be guaranteed to work. The time to locate the item using a linear search in an array of N items has an average value proportional to N/2. If there are few items to be searched through, N/2 may be perfectly acceptable, but as the number of items increases, the time taken to perform the search may become excessive.

If the data in the file is known to be in a sorted state, however, there's a far more efficient searching method, known as the 'binary search', which works in the following way. Suppose you want to find the definition of the word 'leptodactylous' in a dictionary. You don't start at the first page and see if it's there, and go on to the second page if it's not, working your way through the dictionary until you find it. Instead, you put your thumb roughly in the middle of the book, open the page and see what's there. If the page you open happens to start with 'metatarsal', you know you've gone too far, so the second half of the book is irrelevant and the word you want will be somewhere in the first half of the book. You then repeat the process, treating the page you originally opened as though it were the end of the dictionary. Again you split the first part of the dictionary in two and open the page to find 'dolabriform'. This time you know that the page selected is too 'low' and (for the purposes of our search for 'leptodactylous') can be considered as though it were the first page — all earlier pages are irrelevant as they are known to be too 'low' in the sense that 'l' is 'higher' than 'd'. The 'first' and 'last' pages of the dictionary can now be considered as the ones starting with 'dolabriform' and 'metatarsal' respectively. Again you put your thumb in the middle of the 'relevant' section and open up at 'ketogenesis'. Again this is too 'low' so the word we are looking for must lie between this page and the 'metatarsal' page. Repeating this process often enough is guaranteed to locate the word we are looking for — as long as it is in the dictionary!

In the example we have just considered, 'leptodactylous' was the 'search key'. The search key is the entry we are trying to find. Each time we examine a record, we will compare the search key against the 'record key' to locate the 'target' or 'victim'. Together with the record key we can expect to find what is called 'additional information', logically enough. The additional information for the record key 'leptodactylous' would be the dictionary definition of the word — in this case, slender-toed.

The analogy with searching through a file in a database for a target record is a close one, provided that the records have been previously sorted as the entries in a dictionary have. Think how difficult a dictionary would be to use if the entries were in the order the lexicographer first thought of them!

The search routine required for our address book will need to be more complicated than we might first appreciate for reasons that will become apparent. The first thing the search routine — let's call it *SCHREC* for the time being — will do is request the name to be searched for. This is called the search key. Suppose that somewhere in the file there is a record for a person called Peter Jones. The record for this person will have a field (with the name in standardised form) containing JONES PETER. The search routine might prompt us with a message such as WHO ARE YOU LOOKING FOR?, and we would respond with PETER JONES, or perhaps P. JONES or Pete Jones. Before this gets too complicated, let's assume that we respond with the full name, Peter Jones. The first thing the search routine will do will be to convert this response to the standardised form, JONES PETER. Next, it will compare our input, the search key, with the various contents of the MODNAM$ fields. If the program were using a linear search, the search key would be compared with each MODNAM$ field in sequence until a match was found or until it was discovered that an exact match did not exist.