Changing Places

After looking at how to insert new records, we move on to ways of retrieving them. As anticipated, we first encounter the problem of finding an exact match

We ended the last instalment with an exercise for you to write a database-type program that allowed data to be entered into it. Let's look at some of the steps involved in entering a new record as a way of continuing our examination of what is involved in the INITIALISE stage of our main program. First, let's assume that there are the following fields and corresponding arrays:

FIELD	ARRAY
1 NAME field	NAMFLD$
2 MODIFIED NAME field	MODFLD$
3 STREET field	STRFLD$
4 TOWN field	TWNFLD$
5 COUNTY field	CNTFLD$
6 PHONE NUMBER field	TELFLD$
7 INDEX field	NDXFLD$

The meaning of most of these fields should be reasonably clear, with the possible exception of fields 2 and 7. Let's first consider the MODIFIED NAME field. When we initially looked at the problem of the data format for the name, we debated whether to have the name format tightly specified (rigid) or loosely specified (fuzzy) and we opted for the latter. Since the way a name can be entered is extremely variable, a rigid format would have made search and sort routines very difficult. To solve this we decided that all names would be converted to a standardised format: all letters converted to upper case, all non-alphabetic characters (such as spaces, full stops, apostrophes, etc.) removed and that there would be only a single space between the forename (if any) and the surname.

The need to standardise names like this arises because the sort and search routines have to have some way of comparing like with like. On the other hand, when we retrieve a name and address from the database, we want to have the data presented in the form it was originally entered. There are two ways of handling this problem: either each name filed is converted into standard form only when sorts and searches are taking place, or the name field can be converted into standard form and stored as a separate field so that sort and search routines can have instant access to standardised names.

There are advantages and disadvantages in both approaches. Converting the name fields temporarily when they are wanted by other routines saves memory space, since less data needs to be stored in the file. On the other hand, this procedure is extremely time-consuming. However, if a separate field is reserved for the standardised form of the name, the conversion will need to be performed only once for each record. And although extra memory is consumed, searches and sorts will be executed quicker.

The other field that may cause confusion is the INDEX field. This is really included as a spare field to allow for future expansion or modification of the database without the need for major rewriting of the program. Its inclusion introduces the topic of 'binding' — a term that means the fixing of data and processing relationships. All the fields or elements in each of the records are bound because they have the same index (the same element number or subscript in their respective arrays), and because all the fields in a record will be stored in a file together. This can make the addition of new data types or relationships at a later stage a difficult task, possibly involving the complete reorganisation of the file structure and a major re-writing of the program. The incorporation of the INDEX field at this stage will make future changes to the program much simpler.

Before attempting to add a new record to the database, we will make a few assumptions about the structure of the files. First, we will limit the number of records to 50 (even though this is really too small for a useful address book — we'll find out how to handle large amounts of data later). We will also assume that all the data has already been transferred — as part of the INITIALISE procedure — into arrays.

When a new record is added, it is simplest to add it to the end of the file (that is, to the first empty element in each array). There is a good chance that the new record will be out of order with the others, but that is a problem we can investigate later. The first thing to do, therefore, will be to find out how big the array is. Since this is a piece of information likely to be useful in many parts of the program, the best place to do it is in INITIALISE. This is a clear case of the need for a global variable (that is, a variable that can be used in any part of the program). We will call it SIZE. Another global variable likely to be useful is the index of the current record. Since no record will be current when the program is first run, assigning an initial value to CURR will have to wait until the program does something to the data. CURR can, however, be initialised to zero in the INITIALISATION procedure. Initialising a variable to zero is not strictly necessary in BASIC as this is done automatically. It is, however, good practice and

TONY LODGE