Time And Motion

Sorting an array in Basic can be a time-consuming operation, but will ultimately speed up our searches for specific records

So far we have developed most of the code needed to create entries in the address book 'database', but have not yet tackled the necessary programming for saving the entries on tape or disk. The only major omission has been a suitable routine for the creation of the MODFLD$ field, as specified earlier in the course.

The complete program to do this is given in this instalment of the course. First, all characters are converted to upper case (capital letters) in lines 10250 to 10330. Lines 10350 to 10370 then count through the characters in the string and check each one to see whether it is a space. The last space encountered leaves the variable S set to the value corresponding to its position in the string.

Lines 10400 to 10420 transfer characters, one at a time, from the string of upper case characters to CNAM$. Characters are transferred, until we get to the last space, if they have an ASCII value of more than 64. Any characters that fail this test are ignored, so this process eliminates full stops (ASCII 46), apostrophes (ASCII 39), spaces (ASCII 32) and all other punctuation marks. Lines 10450 to 10470 do the same for the characters after the final space, transferring them to SNAM$.

If N$ contains only a single word, TREVANIAN, for example, variable S will be 0 and all the characters will be transferred to SNAM$. The variable used for the forename has been called CNAM$ rather than FNAM$. CNAM$ is used to remind us of 'Christian name', as variables starting with the letters 'FN' will confuse many BASICS into thinking that a call to a user-defined function has been made.

Lines 10490 and 10500 are needed to set the string variables used in this routine to nulls before they are used again. This is a point to watch out for whenever structures of the type LET X$ = X$ + Y$ are used. Failure to 'clear' the variables will result in more and more unwanted characters accumulating in them each time they are used. Notice that CHOI is set to 0 in the ADDREC routine, since we only want to make sure that the user adds a record if there are none in the file (that is, the first time the program is used).

Now that we have a way of adding as many new records to the file as we want, we need a way of saving the file on tape or disk. The simplest way would be to write all the records to the data file (ADBK.DAT in this version of the program) in the order they happen to be in. The chief disadvantage of this approach is seen when we need to search the file for a particular record. If we cannot be sure that all the records in the file are sorted in some way, the only way to search for a record would be to start at the beginning and examine each record in turn to see if it matches the 'key' of the search. If the record you were searching for happened to be the last one entered, every record in the database would need to be examined before the one you wanted was located. If the last record entered was for a William Brown (i.e. MODFLD$(SIZE-1)="BROWN WILLIAM"), a search routine should anticipate the record to be somewhere near the beginning of the file — if the records had been sorted. Unfortunately, both sorting and searching are very time-consuming activities; so it is a question of determining your priority. We have adopted the principle that an address book is consulted far more often than it is added to (or modified in some other way). This being so, it is better to assume that searches will be far more frequent than sorts, so we will always ensure that the records are sorted before they are stored in the data file after the program has been used.

With this in mind, a variable called RMOD is created to use as a flag. It can have one of two values: 0 or 1. It is initially set to 0 to indicate that no record has been modified during the current execution of the program. Any operation that does modify the file in any way — such as adding a new record — sets RMOD to 1. Operations that 'need to know' if the file has been modified will check the value of RMOD before proceeding. For example, EXPROG, the routine that saves the file and exits from the program, checks RMOD in line 11050. If RMOD=0, no sorting and saving is needed as the data file on tape or disk is assumed to be in a fully sorted and unmodified form. Other routines, such as those that search through the file for a particular record, will also need to check RMOD. If RMOD is 0, the search (or other operation) can proceed. If RMOD is 1, the routine will first have to call the sort routine. After the whole file has been sorted, the sort routine will then reset RMOD to 0.

Our sorting routine, called *SRTREC* in the program listing, resets RMOD to 0 in line 11320 after all the records have been sorted. Before going on to look at *EXPROG* (the routine that saves the file on tape or disk and then ENDs), a few words about *SRTREC* are called for. *SRTREC* is a form of a simple sorting technique called a 'bubble sort' (see page 286). There are many ways of sorting data and the bubble sort is one of the simplest and slowest. A good case could be made for a more