Alphasort — a program to sort alphabetical information

Arrange your printout with this program and never lose track of that lady's phone number again!

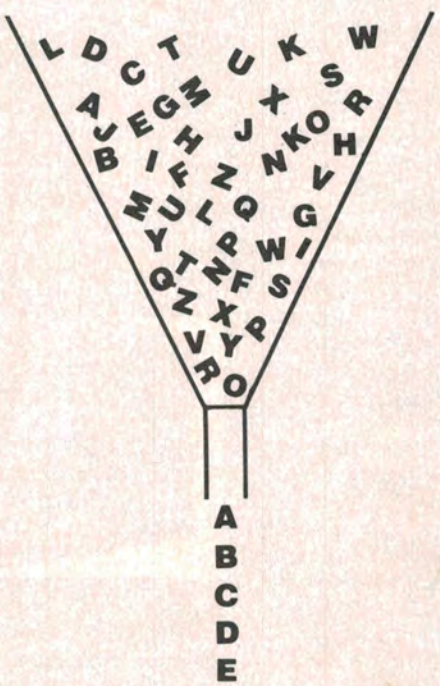
R.A. Jones

THE FOLLOWING piece of software has been designed to fill a number of needs. Although it is a 'stand-alone' program it can be easily adapted to act as a subroutine to fit into other programs, or even turned into a standard utility package. The sole function of the program is to sort lists of names, or indeed any alphabetical information, into order.

Program function

The software relies on the string handling facilities present in most versions of BASIC and without these cannot function as written. Indeed, if these functions are not present any sort program will run so slowly that the user will probably expire from boredom! The ability of these versions of BASIC to use mathematical operators such as $>, <, =$ and $\neq (<>)$ on string functions makes life very easy for the programmer.

The system of sorting is known as a 'bubble' sort (for no better reason than the similarity between bubbles rising through a liquid and the bigger entries rising through the list). It sets no records for speed but it does work and is simple to understand, a feature often worth far more. The two main segments are illustrated in Figure 1 and Figure 2. These are the input routine and the bubble sort routine and are further described later. The full program listing is divided up with REM statements; each of these segments represents a complete entity and can be amended or altered as desired. Some suggestions are given later in this article.



How it works

As previously mentioned, the application of mathematical operators is crucial to the bubble sort. The BASIC allows us to simply compare two string variables and make a decision as to whether one is bigger than the other, or whether they are equal in size. These comparisons are not confined to the first letter but work their way through the entire length of the string; for example:

Given two strings, A$ and B$, we can say that if A$ = "A" and B$ = "B", then A$<B$ is true. Similarily we can compare the string "JONES B G" with the string "JONES B H" and find that the first is 'smaller' than the second.

Given this facility we can sort any stored list of strings into order, either ascending or descending, although the latter is more common (lists of names usually go from A to Z). This segment is illustrated in Figure 1 and is the section of the program tagged BUBBLE SORT.

The first statement simply sets up two variables, one counter and one marker. The variable S is a 'swap' marker and tells us that a change has taken place in the list, the counter T is one less than the number of entries because you can't compare the bottom entry with anything! You now start a loop going for this many counts.

For each entry in the list (array A$(n)) you compare the absolute value with the entry directly below it in the list; if the first is bigger you swap them over and set the swap marker, if not you try the next pair. The changing over is done by the laborious method of putting the larger string into a spare variable, replacing it in the list with the smaller and then putting the larger one back. Anyone using a disk-based BASIC with extra functions can use the marvellous SWAP command and do the whole thing in one go. Having gone through the list once, the whole process is repeated until no swaps are recorded; the sorting process is now complete.

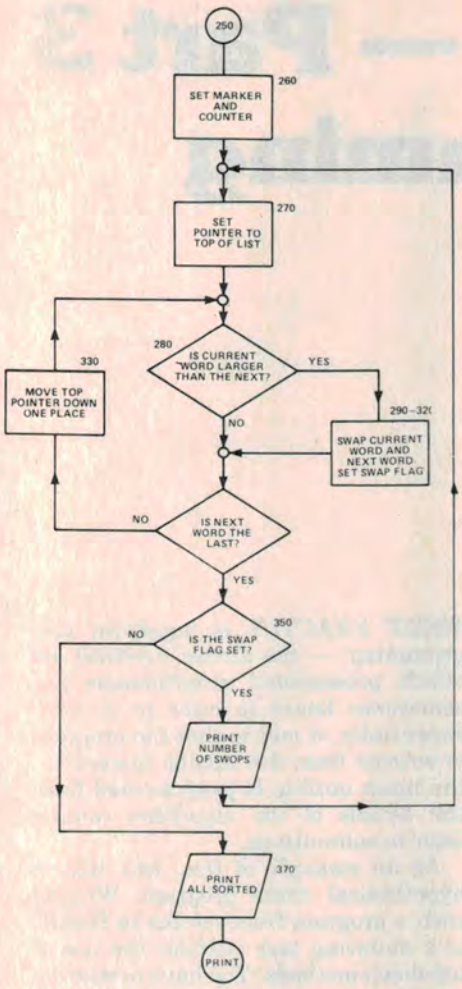


Figure 1. **The routine for bubble sorting strings**

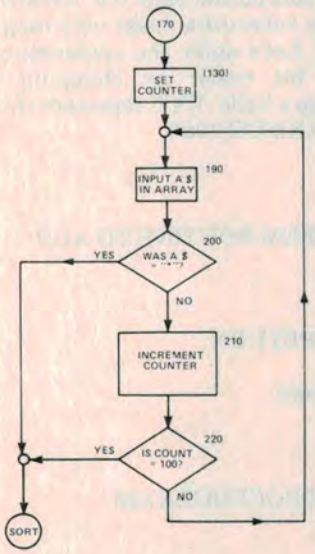


Figure 2. **The input segment in greater detail**

The input stage is also worthy of closer investigation. The maximum number of list entries is set up as 100 but this is really dependent on the amount of memory you have available. As each entry is input from the keyboard it is stored in an array at a position corresponding to its entry point. It is worth noting that the array starts at 0, a location which is often ignored or even forgotten. Entries continue until "*" is found, which terminates the routine. We now have an array full of raw data and a counter which tells us how many entries there are in the array. We may now sort it.

Getting listed

Actually producing the final list is dead easy; you simply output the array element by element. However, if your list is longer than your screen has lines, you may like to implement a loop which outputs a set number of entries at a time. A routine is given in the program called LINE LOOP, which does just this. The required number of lines is input to the program and then the routine waits for any key to be hit before outputting the first batch.

Enhancements

Some obvious goodies that can be built in are: reading data from a file, outputting to another file, outputting to a printer and doubtless others of a more specialised nature. Taking the first and second items, it should not prove too difficult to open a file and read entries both from it and back to it instead of keying them in by hand. Commands such as OPEN, INPUT# and PRINT# should be recognisable to most systems running a reasonable BASIC.

Printing out lists is also a matter of calling the printer rather than the VDU; if your system supports LPRINT then life is simple indeed! All you really need to do is to call a response from the keyboard to direct the output to the required device; it is worth making life idiot-proof by having the VDU as the default option. Owners of systems such as the PET who are using interfaces to connect to printers will have to treat the output like a file, but you must remember to CLOSE it after output is complete or else all your screen prompts tend to end up in the middle of your listing.

Other possibilities for the program are multiple lists. These offer no serious difficulty; you merely choose which list you are going to sort on and then, as you swap on the chosen list, swap the others as well. It is in situations such as this that the time taken starts to mount up. If we take a sample list such as fred, john, ian, bert, harry, the following swaps take place:

fred,john,ian,bert,harry
fred,ian,john,bert,harry
fred,ian,bert,john,harry
fred,bert,ian,john,harry
bert,fred,ian,john,harry
bert,fred,ian,harry,john
bert,fred,harry,ian,john

Now, if we had a parallel list of, say, their ages, the swap time would have been almost doubled. The maximum number of swaps that can take place is the factorial of the number of items in the list, the actual time taken being rather machine-dependent for obvious reasons. This time will also increase in direct proportion to the number of 'columns' that you have. As mentioned earlier, the program makes no apologies for its lack of speed. It is, however, as near universal as possible. ●

```
100 REM**ALPHASORT 2
110 REM**INITIALISATION
120 PRINT"[CLS]":CLR
130 DIM A$(100):EN=100:CT=0
140 PRINT "PLEASE INPUT NAMES, WHEN YOU
    ARE"
150 PRINT "READY TO SORT TYPE '*' "
160 PRINT
170 REM**INPUT ROUTINE
180 PRINT "YOU HAVE ROOM FOR ";EN;" MORE
    ENTRIES."
190 INPUT A$(CT)
200 IF A$(CT)="*" THEN 250
210 CT=CT+1:PRINT"[CLS]"
220 IF CT>99 THEN 250
230 EN=100-CT:GOTO 180
240 END
250 REM**BUBBLE SORT
260 S=0:T=CT-1
270 FOR L=0 TO T
280 IF A$(L)<=A$(L+1) THEN 330
290 S$=A$(L)
300 A$(L)=A$(L+1)
310 A$(L+1)=S$
320 S=S+1
330 NEXT L
340 PRINT "[CLS]";S;" SWAPS OCCCURRED"
350 IF S>=1 THEN 260
360 PRINT
370 PRINT "ALL SORTED !"
380 REM**SIMPLE OUTPUT ROUTINE
390 PRINT
400 PRINT "HIT ANY KEY TO LIST"
410 GET R$:IF R$="" THEN 410
420 PRINT "[CLS]"
430 FOR LP=0 TO CT
440 PRINT A$(LP)
450 NEXT LP
460 END
470 REM**LINE LOOP OUTPUT
480 PRINT
490 PRINT "HOW MANY LINES ON YOUR VDU";
500 INPUT SL
510 SL=SL-1:LP=0
520 FOR P=LP TO LP+SL
530 PRINT A$(P)
540 NEXT P
550 PRINT "HIT ANY KEY TO CONTINUE"
560 PRINT " '$' WILL BREAK."
570 GET K$:IF K$="" THEN 570
580 IF K$="$" THEN END
590 IF CT-LP<SL THEN 520
600 SL=CT-LP
610 GOTO 520
620 END
```

The complete program listing; see the text for suggested enhancements.