Assignment #2

Due Date: Tuesday, 28 July 1992

15 July 1992

You will write a program which sorts the lines in its input into alphabetical order.

1 Requirements

Your program will read lines of text from the input source. Each line will be separated from the next by a newline character, which is not part of the line. Your program will print out the input lines, but in alphabetical order.

If the input contains many lines, you do not have to read and output them all, but you must process at least the first 500 lines in the input. If a line is very long, you may truncate it, but you must save at least the first 80 characters of each line.

Your program will examine its command-line arguments to see where the input comes from and where the output must go to: The first argument will name an input file, the second will name an output file. If one or both arguments are missing, the input or output will default to stdin and stdout, respectively. Your program must recover gracefully from incorrect arguments, printing appropriate messages if (for example) the user supplies too many arguments or names a file that does not exist.

2 What to Hand in

You should hand in a disk with your source code and an executable, and any special instructions to me. You may choose to supply a log file that demonstrates your program, if you like. I will not accept paper copies of solutions to this assignment.

3 Assignment-Specific Points

Seven points for printing a correct sorted list. Two points each for handling very long lists and for handling long lines correctly. Two points for allowing command-line arguments to specify input and output sources. One point for recovering gracefully from argument errors.

4 About Sorting

The **requirements** section says that you have to print out the input lines out in alphabetical order, and to do that you will need to *sort* them. To *sort* data means to place it in some order, alphabetical in this case. The sorting is the big deal on this project. All the other stuff is just decoration.

There are a vast number of different algorithms for sorting data. Some are easier to understand than others, and some require language features we don't have yet. Since I need to show you an algorithm that is easy to understand and that is also possible to implement with what we know now, I have a choice of *straight insertion* or *bubble* sorting.

4.1 The Straight Insertion Sort

The straight insertion sort is familiar to you already: It's how you sort a deck of cards into order. To sort a deck of cards, you keep two piles: One pile initially has all the cards in it; it's called the *source pile*. The other pile is initially empty; that's the destination pile. To sort the deck, hunt through the source pile until you find the lowest card—say 2♣. Take that card out and put it in the destination pile. Now find the lowest card that's left in the source pile—that'll be 3♣. Take it out and put it on the destination pile. Continue in this way until you've taken the last card, the A♠, from the source pile and put it on the destination pile. Now the destination pile is sorted.

To do this with elements of an array is not much different. You have a source array, which initially contains the data you want to sort, and a destination array of the same size. Find the smallest element in the source array. Move it to the first empty space in the destination array; this involves copying the element from the source to the destination, and also obliterating it in the source so that you don't find it again next time. Repeat this until all the elements are moved from the source to the destination; at this point all the source elements have been obliterated, but copies are in the destination array in sorted order.

4.2 The Bubble Sort

The bubble sort is often maligned and a little less familiar than the straight insertion sort, but it's really no worse, and it has the advantage that it doesn't require an extra destination array.

Lay out a dime, a nickel, a quarter, a penny and a bean, in that order, on the table, and follow along with the description. We'll sort these things into order by value, so that the bean, which is worthless, is on the left, and the quarter is on the right.

Compare the leftmost two coins, the first and second ones. If they're in the correct order, do nothing. Otherwise, swap their positions. Then compare the second and third coins and switch them if they're out of order. Continue doing this until you compare the last two coins, which should be the quarter and the bean.

Each time we compared two coins, the greater one wound up on the right and we used it in the next comparison. Somewhere along the line we compared something with the quarter, and the quarter appeared in all the comparisons after that; it kept 'bubbling' over to the right. The quarter is now in the rightmost position of the line of coins, which is where it should be.

Now repeat the process. The dime will bubble right to the correct position. Repeat it again, and the nickel will move to the correct position. Each time we run through the coins, at least one coin will bubble into the correct position. Therefore, if there are n coins, we needn't repeat the bubbling process more than $n - 1$ times.

4.3 Stupid-Sort

To stupid-sort a pack of cards, throw the cards down the stairs and then gather them up again. Examine the pack to see if the cards are in order. If they are, stop. Otherwise, repeat the process.

Stupid-sort is not a reasonable sorting algorithm, because you have to throw the cards down the stairs about $8 \cdot 10^{67}$ times before it works.

4.4 What Sort to Use

You can use any reasonable sorting algorithm you like, including the bubble sort or the straight insertion sort. You may not use stupid-sort. If you want to use some algorithm not discussed here, it would probably be a good idea to discuss it with me first, but you don't have to.