Lab 7: Sorting

Goals

Build a program that can perform a simple exchange sort. The program will read an arbitrary collection of integers into an array and then sort them into ascending order.

Background

Sorting

- Sorting is a very common and useful operation in computing. A sorting procedure will take a set of values in arbitrary order and put them into either ascending or descending order.
- There are many well known sorting algorithms. The faster ones are more complicated to implement than the slower ones.
- One of the simplest sorting algorithms is a form of exchange sort that is commonly called the *bubblesort*.
- The basic strategy on which it relies is to swap adjacent elements that are out of order until all values are in order.
- Codelab Unit-T21 of assignment JMW-Assn2 is an example of a program that makes a single pass over an array and swaps adjacent elements of an array if and only if they are out of order.
- Although this program *does not sort* the array it can be used as a basis for a sort.
- What this program *does do is move the largest value that is out of order into the proper location.*
- Thus if an array were to have 50 elements, and we were to apply the strategy of Unit-T21 50 times, then we would be guaranteed that the array would end up with no elements out of order.
- The objective of this lab will be to use this approach to implement a correct sort.

Assignment:

Write a program called lab7.c that contains two functions as follows:

1. Create a main() function that contains an array of capacity 100 ints: `int array[100];` and a variable `int counter;` that will contain the actual number of integers that are read from the standard input before end of file is reached.

   Your *main()* function should use a *while()* loop to read integer values from the standard input into adjacent elements in the array and set counter to the total number of integers read.

   In a *separate* *while()* loop after the read loop completes print all of the values using the "%d\n" format code.

   When this much is working show your TA that you are ready to move to the second step.

2. Create a function `int swapper(int array[], int count)` that is passed an integer array along with the *actual number* of values in the array.

   The function should make a single pass over the array swapping adjacent pairs that are out of order as done in the second loop in the solution to Unit-T21.

   This function should *return the number of actual swaps* (not comparisons) it performed. This function MUST NOT contain any instance `fscanf()` or `fprintf()`.

3. Now return to the *main()* function insert a third *while()* loop *after* the reading loop and *before* the printing loop.

   Like the print loop it should iterate exactly *counter* times.

   In the body of this *while()* loop, simply invoke the *swapper()* function passing it `&array[0]` and `counter`.

4. Extra credit(1): Passing counter every time actually does more work than is necessary in swapper. After the first call to swapper the largest element is guaranteed to by in array[counter – 1]. After the second call to swapper, the last two elements of the array will be correct. Eliminate the unnecessary work by passing decreasing values to swapper().

5. Extra credit(2): Suppose the integers are already in sorted order at the time they are read in. Then there is nothing to do but our program will still call *swapper()* *counter* times. If you ever call *swapper()* and *it doesn't swap anything* then the values are in the proper order and *the sort is finished*. Modify the loop so that it will terminate if swapper() doesn't swap any values.

Turn In Work

Show your TA that you completed the assignment. Then turn in your lab7.c program using the command:

sendlab.101.*section_number* 7 lab7.c