Homework 08

Due Date: Monday 07 April 2014 11:59 PM MST
Note: If you submit after the due date (but before the hard deadline), your submission score will be penalized by 20%.

Hard Deadline: Wednesday 09 April 2014 11:59 PM MST
Note: If you submit any time after the hard deadline, you will not receive credit.

Problem 01 (15 points)

Make your own version of the library function **strcmp(s1, s2)**, which compares two strings and returns –1 if s1 comes first alphabetically, 0 if s1 and s2 are the same, and 1 if s2 comes first alphabetically. Call your function **compstr()**. It should take two **char*** strings as arguments, compare them character by character, and return an int. Write a **main()** program to test the function with different combinations of strings. Use pointer notation throughout.

Problem 02 (15 points)

In this section, we define a sorting program to demonstrate passing arrays and individual array elements using pointers. We use the **selection sort** algorithm, which is an easy-to-program, but unfortunately inefficient, sorting algorithm. The first iteration of the algorithm selects the smallest element in the array and swaps it with the first element. The second iteration selects the second-smallest element (which is the smallest element of the remaining elements) and swaps it with the second element. The algorithm continues until the last iteration selects the second-largest element and swaps it with the second-to-last index, leaving the largest element in the last index. After the *i*th iteration, the smallest *i* items of the array will be sorted into increasing order in the first *i* elements of the array.

As an example, consider the array

```
34   56   4   10   77   51   93   30   5   52
```

A program that implements the selection sort first determines the smallest value (4) in the array, which is contained in element 2. The program swaps the 4 with the value in element 0 (34), resulting in

```
4   56   34   10   77   51   93   30   5   52
```

The program then determines the smallest value of the remaining elements (all elements except 4), which is 5, contained in element 8. The program swaps the 5 with the 56 in element 1, resulting in

```
   4   5   34   10   77   51   93   30   56   52
```

On the third iteration, the program determines the next smallest value, 10, and swaps it with the value in element 2 (34).

```
   4   5   10   34   77   51   93   30   56   52
```

The process continues until the array is fully sorted.

```
   4   5   10   30   34   51   52   56   77   93
```

After the first iteration, the smallest element is in the first position. After the second iteration, the two smallest elements are in order in the first two positions. After the third iteration, the three smallest elements are in order in the first three positions.

The given program, **Homework08P02.cpp**, implements selection sort using functions **selectionSort** and **swap**. Function **selectionSort** (lines 32–49) sorts the array. Line 34 declares the variable smallest, which will store the index of the smallest element in the remaining array. Lines 37–48 loop size - 1 times. Line 39 sets the smallest element's index to the current index. Lines 42–45 loop over the remaining array elements. For each element, line 44 compares its value to the value of the smallest element. If the current element is smaller than the smallest element, line 45 assigns the current element's index to smallest. When this loop finishes, smallest will contain the index of the smallest element in the remaining array. What you need to do are listed as follows:

- On line 47 calls function swap (lines 53–58) to place the smallest remaining element in the next spot in the array (i.e., exchange the array elements array[i] and array[smallest]).
- Starting from line 54, implement the function swap using the given parameters.