CISC 1110 Assignment 6 Spring, 2017 Due May 1, 2017

Write a complete C++ program to read in, sort, and perform other statistical actions on a set of data. The program will read from an input file and write all results to an output file.

- From an input file, the program will read a set of student's 3-digit id numbers and their SAT scores.

- The program will sort the id numbers into descending numerical order, being sure to carry along the corresponding SAT scores. The program will then print the sorted lists, giving both id numbers and SAT scores.

- The program will then sort the SAT scores into ascending numerical order, carrying along the corresponding id numbers. The program will print the sorted lists, giving both id numbers and scores.

- Then the program will call functions to find the max and min of the SAT scores and the mean of the SAT scores.

- Finally, the program will read in two scores and call a function **search** once for each value, to see if the score is found in the array scores.

Here are the details:

The main program will declare two arrays (call these arrays ID and scores or something similar) and an integer variable n.

1. The main program will send the two arrays and the reference parameter n to the **readdata** function which will read all the data from a file; it will read in n and then read in n data values in parallel into the ID array and the scores array. The data set consists of groups of data, each of which contains a person's 3-digit id number and an SAT score (a number in the range 600 to 2400). Two sample sets of data might be 123 1850 and 456 1260. The arrays and n will have values upon return to main.

The main program will call **printarrs** to print the original set of data, in columns next to one another in the form of a neat table, with a heading at the top, column headings, etc.

2. Then the main program will send the array of id numbers, the array of scores, and the size n to a sorting function called **lsort (that's an "L", not a one)**. This function will sort the id numbers into descending numerical order, being sure to maintain the link-up of id numbers and scores (see warning below). This function will use a linear (selection) sort to sort the arrays.

When **lsort** finishes and returns control to the main program, the main program should call a printing function **printarrs** to print the two arrays in columns next to one another. Before printing the arrays, print an overall heading (like "Sorted by ID in Descending Order", plus headings for the id numbers and scores columns.

Warning: Be sure to maintain the match-up of id numbers and scores. For example, 456 should always be associated with 960, no matter where 456 moves in numerical order; similarly, 123 should stay with 1450.

3. Next the main program will send the same three parameters to a second sorting function, **bsort**, which will sort the SAT scores into ascending numerical order, being sure to maintain the link-up of id numbers and scores. This function will use the bubble sort to sort the arrays.

When **bsort** finishes and returns control to the main program, the main program should call **printarrs** to print the two arrays. Before printing the arrays, print an overall heading, plus headings for the id numbers and scores columns.

4. Next the main program will call a function **mean** which will find the mean of the SAT scores. (The mean is just another name for the average, so this is not new.) The **mean** function receives two parameters, the array scores, and n, the number of values in the array. The function will return the mean, which should be a double. The main program will print the mean, together with a message.

5. Next the main program will call a function **findlimits** which will find the highest SAT score and the lowest SAT score and return both to the main program using reference parameters. The **findlimits** function receives four parameters: the array scores, the number of values in the array n, and reference parameters max and min. After returning to main, main should print the high and low scores together with messages.

Note that this is extremely easy once the score array has been sorted. THINK before you write the function. If you loop through the entire array twice (or even once), I will ask you to redo the program.

6. Next the main program will read in a score value from the same input file and call a function **search** to find the value in the array. The main program will read in a second score value from the flle and also call **search** to find the value in the array. The function will receive three parameters: the array scores, the value n, and the value to find in the array. The function will return the position of the value in the array or -1 if the value is not found. **After each call, the main program will print the score value and the matching ID or else the score value and the message "not found."** One of the values you search for should be present in the array; the other should NOT be found in the array, so that you test all lines of code.