Santa Monica College
CS 20A, Fall 2014

Programming Project 3 - Due: **Wednesday**, November 5 at **6:45 PM**

Satisfied with the work you did finding SMC alumni at UC schools, your boss has come to ask for your help again.

She has a large file of UC students already sorted by ID. It would be helpful to her to have the students sorted by the school they are attending, and students attending the same school sorted by ID in ascending order. For example:

Joe Paarmann, 3,UCB
Otha Baloy,5,UCB
Alton Hamblet,1,UCD
Jessie Merle,7,UCD
Lawanda Doell,9,UCD
Alva Zajicek,4,UCI
Chester Kemfort,2,UCLA
Alice Mines,6,UCLA
…

She tried sorting the students herself but when she sorts by school the students are no longer sorted by ID within each school and vice-versa. Having learned of stable sorting you agree to take on this simple task.

Part 1:

The students in the file are already sorted by ID so if you use a stable sort you can get the students sorted by school and by ID within each school.

You plan to use the Standard Template Library sort function. The sort function we used in project 1 used the *natural ordering* of students to do the comparison (*the implementation of operator < in the student class*). The STL sort function also allows the client to provide a comparison function that compares two elements by any criteria you may choose to. You will implement the given function `bool compareBySchoolName(Student s1, Student s2)` which compares two students by school name (the function should return true **if and only if** s1's school comes before (is less than) s2's school) and use that in the sort function like below.

```
sort(students, students + len, compareBySchoolName);
```

You try that out and you find that the file is now sorted by school name, but the ID order is ruined. After some thinking and searching you conclude that the sort function of the STL gives no guarantees of stability. If you want stability you have to use the stable_sort function. You just change the line above to the one below and you are happy with the results.

```
stable_sort(students, students + len, compareBySchoolName);
```

Part 2:

Part 1 did what you wanted in 3 lines of code but you notice that there are only 7 UC schools in the whole file and you think a non comparison based sort would be faster. You decide to implement your own sorting algorithm for the problem based on counting sort. The idea is very simple:

- You do a pass through the student array to count how many students there are in each school. You store the counts in a counts array.
- You allocate another array of Students with size equal to the input array. You will use this temp array to hold the sorted elements.
- Using the counts array you calculate the beginning index for each school in the sorted array

by noticing that the starting index for a school is the cumulative number of students for all the schools before it in the array. E.g.: if the counts are 5, 10, 12 for schools a, b, c then the starting indexes for each school's students in the sorted array are 0, 5, 15.

- You do another pass through the student array and whenever you encounter a student from a school you add that student to the right portion of the helper array and increment that portion's index.
- The helper array is now sorted by school and by ID within each school. Copy the helper array's elements to the input array.

Hints:

All the code not dealing with sorting is that of Project 1 and it has been given to you in Student.h and Student.cpp. A getSchool() method was added that returns the school name. **All you have to do is fill in the empty bodied methods in CS20AProj3.cpp.**

For Part 2 make sure you really understand the algorithm (simple as it is) before you get down to writing any code. Do not write even one line of code if you are not very clear what needs to be done (best achieved by doing a simple example on paper – no code involved at all).

Use a smaller input file until you are sure your code is functionally correct. You can then use the given large file to do the timing.

Deliverables:

You should submit a zip file named project3_first_last.zip (where first and last are your first and last name) containing **ONLY** the **2 files below. You should NOT submit ~~Student.h or Student.cpp.~~**

CS20AProj3.cpp
output.txt // a copy of the program output (from the main function) containing
// the **timing printouts** and the **first 3 and last 3 lines** from **your**
// implementation's uc_by_school_by_id2.txt file.

How you get points:

`sortByGroupById1`	20 points
`sortByGroupById2`	80 points

How you lose points:

- You do not follow the given directions and decide to make changes "for fun". Specifically, **do not change the code given to you**. Can use helper methods if you need to (you shouldn't really need to) but do not change the given code (for the time function, you can change that if you need to make your compiler happy).
- You keep separate counter variables for each school in Part 2. That is probably doable if you have 7 schools but would not be if we had 100 schools for example. **You must use an array for the counts. Your algorithm must be very easily modifiable to support *k* schools.**
- Your algorithm for Part 2 is inefficient. **It must be O(n) time and O(n + numSchools) extra space.**
- You submit your whole workspace. **Submit only the files the project asks for.**
- If any of your code prints anything at all on the console except for the timing output. **Remove all your print outs, debug statements, etc. Clean up your code and do not leave clutter behind.**
- Your code has no comments where needed. **Comment your code appropriately.**