Lab 13: Using Vectors

In this lab you will get practice working with ***vectors***. The project will use a class similar to the `student` class that was created in Lab 10. You will write a main program that uses my versions of the `student.h` and `student.cpp` files.

Exercise

Write a program that creates a `vector` whose base type is the class `student`. The vector is initially empty. It should be filled with the data coming from an input file. The program should have a single argument on the command line that is the name of the input file. For this lab assume that the file contains valid data. Each line consists of a record for a single student and has the following format:

```
last_name first_name id status gpa number_of_credits
```

The last two fields on the line are optional (they are either both there, or both missing). The entries on a single line are separated by any number of space characters, meaning spaces or tabs.

Once the list of students from the file is read into your vector, your program should display a menu that gives the user the following options:

```
[a] Display all records, in alphabetical order by last name.
[g] Display all records sorted in descending order of gpa.
[q] Exit.
```

and repeatedly respond to the user's choice and then redisplay the menu until the user quits. In all cases, when the records are displayed, the fields must be tab-separated. For option `'a'`, if last names are the same, first names should be used to break ties. In option `'g'`, if gpas are the same, any order is acceptable.

Implementation Suggestions

Since the number of entries on a line in the input file is variable, you cannot use the >> stream extraction operator for reading one element at a time. You need to read an entire line and then parse that line. There are a few different ways that you can do this, such as using the global `getline()` function to read into a C++ string, or the `istream::getline()` member function to read into a C string. Once your data is in a string of either kind you have many ways to extract the fields.

If you store into a C string, you could use one of the C string library functions to get the words within the string.

If you store into a C++ string, you could also do this by extracting the C string from it, or you could use one of several `string` class member functions, such as:

`find_first_of()` Find character in `string` (public member function)

`find_last_of()` Find character in `string` from the end (public member function)

`find_first_not_of()` Find absence of character in `string`

`find_last_not_of()` Find absence of character in `string` from the end (public member function)

You can find a more detailed explanation and the exact parameter list on the website for the `string` class. Regardless of which method you choose, remember that the gpa and number of credits either are both present or both absent from the line.

Extra Credit

- If the gpa is present but not the number of credits, catch this. (5%)
- If the gpa or number of credit is not numeric, catch this (5%)
- In the gpa sort, use the last name in increasing alphabetic order as the tie-breaker. (10%)

What to Submit

Submit your program, however complete it is, by the end of today's lab, i.e., before the end of the class at 2:00 P.M. ***There is no grace period for this. Programs submitted after 2:00 PM will not be accepted.*** The instructions for submitting are:

1. Create a directory in
   `/data/biocs/b/student.accounts/cs135_sw/cs136labs/lab13/submissions`
   whose name is your *username*. For example, I would create the directory `sweiss`.
2. Copy your file, which should be named `lab13_main.cpp`, and the `student.h` and `student.cpp` files, to that directory. You will lose 5% of the grade if you misname the file! Make sure that each file has a preamble with your name and other appropriate information in it.
3. Change the permission on the directory that you created so that no one else can read or modify it.

Do not submit executable files. Remember to document your code (both preamble and comments in the code) and make it easy to read. Your work will be graded based on the rubric outlined in the Programming Rules document.