Homework #2

Complete By: Wednesday, January 9th @ 11:59pm
Assignment: completion of following exercise
Policy: Individual work only, late work *is* accepted
Submission: electronic submission via Blackboard

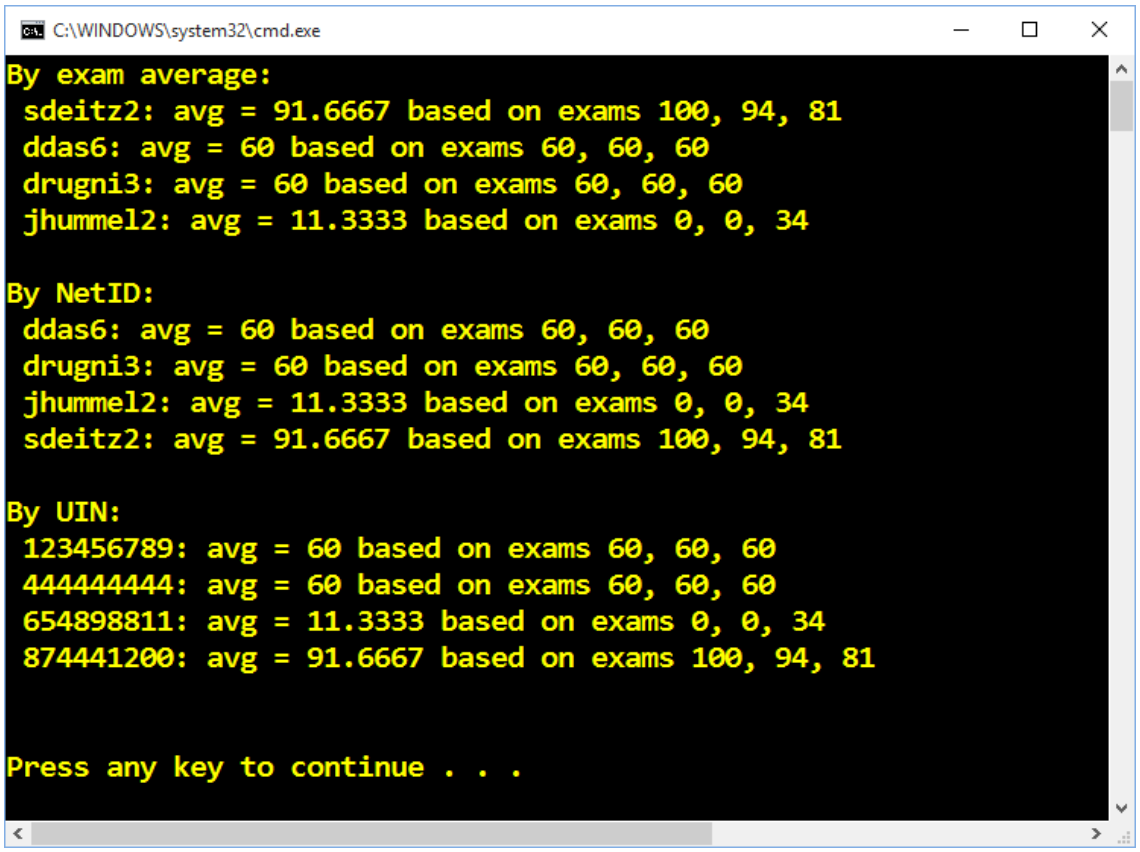
Assignment

Assume the existence of a text file named "students.txt" that contains info about N > 0 students. Each student has a NetID, a UIN, and 3 exam scores; each value is on a line by itself. Here's an example file:

```
drugni3
123456789
60
60
60
jhummel2
654898811
0
0
34
ddas6
444444444
60
60
60
sdeitz2
874441200
100
94
81
```

Your assignment is to read this file, and then output the data in 3 different ways: by exam average (descending), by NetID (ascending), and by UIN (ascending). For example, given the file above, here is what your program should output:

<< screenshot on next page >>



Note that in the first set of output — by exam average — if two students have the same exam average then output those values in order by NetID. Example: "ddas6" and "drugni3" both have an exam average of 60, so "ddas6" is output first. [In sorting terminology, exam average is called the *primary sort field* and netid the *secondary sort field*.]

The assignment is meant to be straightforward. However, the challenge is in *how* you solve it; you are required to solve it in a particular way. We don't want just any solution, but a particular type of solution that emphasizes the modern aspects of C++. In particular, you must satisfy *all* of the following requirements:

1. No index-based loops. You may use range-based for loops, or iterator-based loops.
2. Do not open/close the file yourself. Instead, use an ifstream approach as shown in lecture 08-24. That means no file >> variable, no calls to getline(), and if you need to know whether the file was opened successfully, look at the vector containing the data or compare the stream iterators. Think C++, not C.
3. You must define a class student to store the data about an individual student; for simplicity, define the class above main in the same source file.
4. You must use a vector of student objects, i.e. vector<student>, to store the data about all the students.
5. You must sort and output based on this vector of student objects.
6. You must sort the data using the built-in sort algorithm, using a lambda expression.

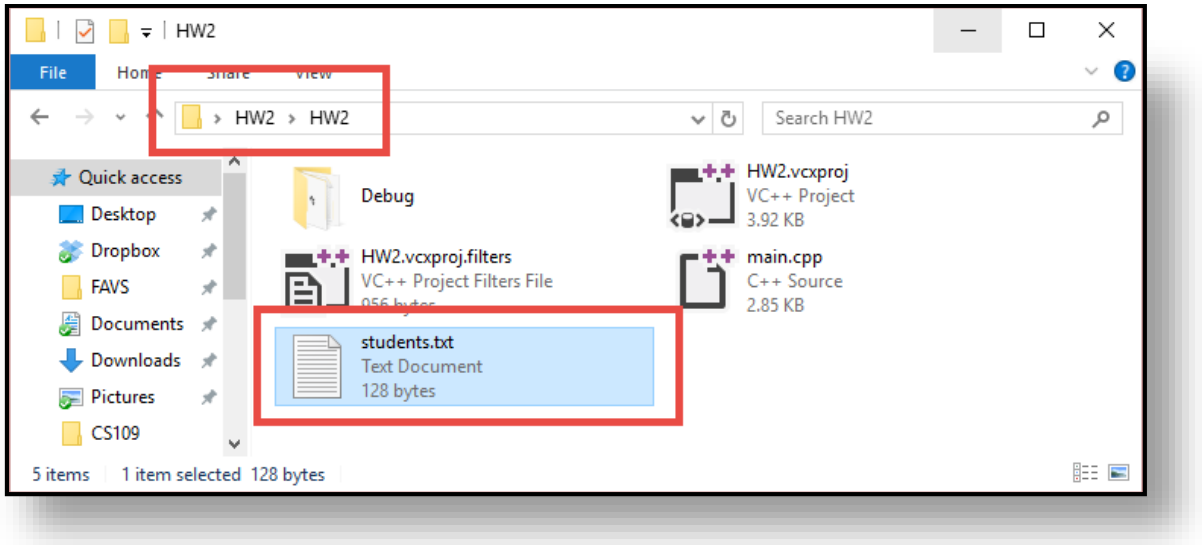
7. Write and call at least one function, passing the vector of students to the function; for simplicity, define the function above main in the same source file.

In short, we don't want a C solution to this assignment, we want a modern C++ solution. Avoid the use of **new** or **malloc**, that's a symptom of C-style thinking. Failure to meet the above requirements means your program will not be graded, and you'll receive a 0 for the assignment.

Assignment Details

Since some of you are still installing Windows and Visual Studio, the use of VS is not required for this assignment. Feel free to use any environment that supports modern C++ (i.e. C++11 or C++14). You cannot use http://ideone.com, since it doesn't support input files. If you have Visual Studio installed (2013 or 2015), I would encourage you to use this IDE. If you need help installing or using Visual Studio, please read the "*Getting Started with Visual Studio*" document on the course web page (under "Misc").

For those using Visual Studio… The input file — in this case "students.txt" — should be placed in the same project sub-folder that contains your "main.cpp" source file. For example, if you name your Solution "HW2", then you will place your input file "students.txt" in the sub-folder "HW2\HW2", like this:



If you store the file in this sub-folder, then in your program you can just refer to the input file by name: "students.txt" — no path is required. *[* ***Note****: don't make the mistake of naming the file "students.txt.txt". If you have file extensions turned* ***on****, then the filename you see should be "students.txt". But if you have file extensions turned* ***off****, then the filename you see should be "students" without the extension.*]

Here are some tips when working with C++… First, to convert a string to an integer, you can use **atoi**, but note that you have to convert the string to a C-style string. For example:

```
string score = "100";
int    exam  = atoi( score.c_str() );
```

Second, to create a student object, I normally do this:

```
student  s(netid, UIN, …);  // create student object:
```

Third, when I'm ready to add this object to my vector of **students**, I then do

```
students.push_back(s);  // add to end of students vector:
```

Finally, since not all C++ compilers fully support the new features of C++11 and C++14, you have to be careful how you initialize class data members that are strings (e.g. NetID). In short, the use of { } doesn't work, so use either one of the following techniques:

```
class student
{
public:
  string NetID;

  student(string netid) : NetID(netid)  // solution 1: use (), not {}
  { }


class student
{
public:
  string NetID;

  student(string netid)
  {
     NetID = netid;  // solution 2: function body assignment
  }
```

Electronic Submission

Using Blackboard (http://uic.blackboard.com/), submit your "main.cpp" source code file under the assignment "HW2". Attach the file, do not copy-paste. If you have multiple files, place all files needed to build your program into a folder, create an archive file (e.g. a .zip), and submit that archive file instead. The top of your main source file should contain a header comment like the following:

```
//
// C++ program to compute student exam averages.
//
// <<YOUR NAME HERE>>
// U. of Illinois, Chicago
// CS341, Fall 2015
// Homework 2
//
```

Your program should be readable with proper indentation and naming conventions; commenting is expected where appropriate. You may submit as many times as you want before the due date, but we grade the last version submitted. This implies that if you submit a version before the due date and then another after the due date, we will grade the version submitted after the due date — we will *not* grade both and then give

you the better grade. We grade the last one submitted. In general, do not submit after the due date unless you had a non-working program before the due date.

Have a question? Use Piazza, not email

As discussed in the syllabus, questions must be posted to our course Piazza site — questions via email will be ignored. Remember the guidelines for using Piazza:

1. *Look before you post* — the main advantage of Piazza is that common questions are already answered, so search for an existing answer before you post. Posts are categorized to help you search, e.g. "Pre-class" or "HW".
2. Post publicly — only post privately when asked to by the staff, or when it's absolutely necessary (e.g. the question is of a personal nature). Private posts defeat the purpose of piazza, which is answering questions to the benefit of everyone.
3. Ask pointed questions — do not post a big chunk of code and then ask "help, please fix this". Staff and other students are willing to help, but we aren't going to type in that chunk of code to find the error. You need to narrow down the problem, and ask a pointed question, e.g. "on the 3rd line I get this error, I don't understand what that means…".
4. Post a screenshot — sometimes a picture captures the essence of your question better than text. Piazza allows the posting of images, so don't hesitate to take a screenshot and post; see http://www.take-a-screenshot.org/ .
5. Don't post your entire answer — you just gave away the answer to the ENTIRE CLASS. Sometimes you will need to post code when asking a question, but then post only the fragment that denotes your question, and omit whatever details you can. If we can't answer your question, we will ask you to post your entire code privately — i.e. "visible to staff only". This is an option when you post.

Policy

Late work *is* accepted. You may submit as late as 24 hours after the deadline for a penalty of 25%. After 24 hours, no submissions will be accepted.

All work is to be done individually — group work is not allowed. While I encourage you to talk to your peers and learn from them (e.g. your "iClicker teammates" or Piazza), this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is described here:

http://www.uic.edu/depts/dos/docs/Student%20Disciplinary%20Policy.pdf .

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. It is also considered academic dishonesty if you click someone else's iClicker with the intent of answering for that

student, whether for a quiz, exam, or class participation. Academic dishonesty is unacceptable, and penalties range from failure to expulsion from the university; cases are handled via the official student conduct process described at http://www.uic.edu/depts/dos/studentconductprocess.shtml .