CSCE 2004 - Homework 2
Due Date - 09/29/2015 at 11:59 PM

0. Overview:

In this assignment, you will develop a program to handle university degree auditing and course management. That is, after taking a set of courses, a student would like to know if she/he has met the graduation requirements for a particular degree (or what the student is missing to meet those requirements). The requirements include required courses, total credit hours, minimum GPA, etc. As a college student, can you see the value of such a tool in the real world?

To build any software system, one must be equipped with the following: a firm grasp of the problem to be solved, the ability of dividing the problem into smaller problems, devising related algorithms to solve the smaller problems and combining their solutions, the ability to translate the high level design into a computer program in a particular programming language (in our case C++), and the ability to debug and test the program.

All of our projects in this course are related to this core problem. So before you begin, you may want to review your own degree requirements to make sure you really understand them. If we are not able to understand a problem, then no matter how good we are as programmers or software engineers, we will not be able to solve that problem.

Achieving the minimum required GPA is one of our fundamental requirements. Our first project was to compute the GPA for five courses. **In this project, we will extend our first project so we can compute the GPA for an arbitrary number of courses requested by the user.**

1. Problem Statement:

The primary goal of this programming assignment is to give students experience with if statements and while loops. A secondary goal is to give students a chance to modify and extend an existing program. A third goal is to practice getting string values containing spaces from the user so we may handle course names such as “Programming Foundations I”. To meet these goals, students will be required to improve their previous GPA program to make the program more flexible and more robust.

Improve flexibility: Your previous program had a hard coded limit of 5 classes that would be used to calculate the GPA. This is very limiting because not all students take 5 classes in a semester, and there is no way to calculate the GPA for a full year, or for the whole degree. To make your program more flexible, you should first ask the user to enter the number of classes they would like to process, and then use a loop in the program to read and process the class grades and credit hours to calculate the GPA.

Improve robustness: In the previous program, you had the option to read grades as the characters A, B, C, D or F. Now, we will add W and I as possible grades (courses with W or I grades will not be used in the GPA calculation). Furthermore, you were not required to any error

checking to make sure the user actually followed directions in the first assignment. In this assignment, you should make sure that the user only enters grades using the A, B, C, D, F, I or W characters. To make your program more robust, you need to add code to check the user's input to see if it is valid and loop until the grade information is input correctly. A similar check should be added to the number of credit hours to make sure these values are sensible.

Enhancement: In addition to the grade and credit hours of a course, we will ask the user to enter the course name (such as "Programming Foundations I") and course number (such as "CSCE 2004") for each course. At the moment, we simply read the above information and gather what we need for GPA computation. In our next project, we will store them in our program so we may list all the courses the user input.

Students are allowed to use any combination of **if** statements and **while** loops they need to complete this assignment. You are **not** required or allowed to look ahead to more advanced C++ features like arrays or user defined functions to perform this task.

2. Design:

For this assignment, you have three big design decisions. First, you must decide what order you want the user to enter input data. Second, you must figure out if the user actually followed directions, and what to do to force the user to enter valid data. Finally, you must implement the GPA calculation with a variable number of class grades. To simply the user interface design task for you and to have a uniform look and feel across students, your program will need to use the same "user interface" as the given executable file. In other words, when your program is executed, it should behave just like the given executable file, except for possible rewording of the instructions to the user. Make sure you write comments in your code to explain your design decisions.

The following further clarifies the user interface to be implemented. Even though many designs are reasonable for the first decision above, we will have **a uniform design** for this project. This allows us to test our program using input redirection and for later file processing. The user will enter the following information one line at a time: 1) number of courses, 2) course name (for example, Programming Foundations I), 3) course number (for example, CSCE 2004), 4) course grade (for example A), 5) course credit hours (for example 4), and repeat 2) to 5 for the rest of the courses. The following is an example of what the user types (program output is not shown):

```
2
Programming Foundations 1
csce2004
A
4
Calculus 1
math2554
B
4
```

3. Implementation:

Since you are starting with your previous program, you should already have something that compiles and runs. Since you must add several features to this program, it is important to make these changes **incrementally** one feature at a time, writing comments, adding code, compiling, and debugging. This way, you always have a program that "does something" even if it is not complete. Also, you should make a copy of your original code, so you can compare the programs later on.

Perhaps you may first add validation code for five courses. After that, you can develop the logic and code for accumulating numerator and denominator of GPA calculation in a loop so we may handle any number of courses. Finally, add the code to read in the course name and course number. (This is merely a suggestion, however.)

4. Testing:

Test your program to check that it operates correctly for all of the requirements listed above. You may use the given executable file to help you. Also check for the error handling capabilities of the code. Try your program with several input values, and save your testing output in text files for inclusion in your project report.

5. Documentation:

When you have completed your C++ program, write a short report (less than one page long) describing what the objectives were, what you did, and the status of the program. Does it work properly for all test cases? Are there any known problems? Save this report in a separate text file to be submitted electronically. Please use the **Project Documentation Template** provided on Moodle as the basis for your documentation.

6. Project Submission:

In this class, we will be using electronic project submission to make sure that all students hand their programming projects and labs on time, and to perform automatic analysis of all programs that are submitted. When you have completed the tasks above go to Blackboard to "upload" your documentation (a single **.pdf**, **.doc**, or **.docx** file), and your C++ program (a single .**cpp** file). Do **NOT** upload an executable version of your program or files in other extensions or format.

The dates on your electronic submission will be used to verify that you met the due date above. **No late projects will be accepted for credit.** Please refer to class syllabus about no late assignment policy as well as suggestions given regarding to submitting a partially working program.

You will receive partial credit for all programs that compile even if they do not meet all program requirements, so please hand projects in on time.

7. Food for thought:

How should we store all the courses (represented by the 4 pieces of data) in our program?

How could we list the courses in alphabetical order according to departmental prefix?

In addition to the 4 pieces of data associated with a course, is there other information or any attribute we should add to a course?