CSCE 2004 - Homework 3
Due Date - 10/14/2016 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 to divide 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.

Project two allowed us to process an arbitrary number of courses that a student has taken and compute the overall GPA. We were able to process the courses, but the courses were not stored inside our program, and we were not able to print the list of courses entered or save the list to a file for future use. In this project, we will address some of these issues.

1. Problem Statement:

The primary goal of this programming assignment is to give students experience with array data structures, if statements, and loops. A secondary goal is to give students a chance to modify and extend an existing program. A third goal is to practice developing a menu driven interface. To meet these goals, students will be required to improve their previous course and course management program to make the program more flexible and more robust.

Enhance and improve flexibility: The program to be developed in this project should maintain all the functionalities such as input validation and GPA calculation of the previous project. In addition, after the list of courses is read in (very similar to hw2), the user is presented with a **menu of choices**. Please run the sample executable (hw3, see instructions on blackboard) provided to understand the expected behavior of your program. Here is the *output* of the executable that shows what the menu could look like:

```
Welcome to the interactive menu-driven part of the GPA and Course storage program.
Please enter the character next to the choice you wish to pick.
Here are your options:
A(a). Compute the GPA of all courses
B(b). List all courses
C(c). Compute the total credit hours of the courses with grade D
D(d). Compute the GPA for a particular semester
E(e). Add another course to the course list
Q(q). Quit the program

Please choose one of the above
```

The input that the user can chooses corresponds to a menu item. Your program should display the **following menu** of choices and prompt the user for one of the menu items. The program should return to the menu after the task has been completed. For example, if the user runs the program, chooses 'B', and views the listing of all courses, the program should then return to the menu so the user may enter another choice. Once the user enters a 'q' or 'Q', the program should terminate. The character input for the menu choices are:

- 'A' or 'a' for computing the GPA of all courses
- 'B' or 'b' for listing all the courses
- 'C' or 'c' for computing the total credit hours of the courses with grade D
- 'D' or 'd' for computing the GPA for a particular semester
- 'E' or 'e' for adding another course to the course list
- 'Q' or 'q' for terminating the program

A course had 4 pieces of information in your previous program: **course name**, **course number**, **grade**, and **credit hours**. In this project, we will add the **semester the course was taken** to the 4 other pieces. This includes a season (**Fall**, **Spring**, or **Summer**) followed by a space, then followed by the year. For example, `Fall 2014`, `Spring 2015`, or `Summer 2015`.

Improve robustness: If a user enters the wrong menu choice, proper feedback should be provided. If a user wants to add a course, and the program does not have room for it, the course will not be added and proper feedback should be provided. In menu option 'D' or 'd', if no course was taken in the semester and year requested, the user should be asked to enter the semester information again (e.g. using input validation loops as discussed in class & lab). Please refer to the sample executable hw3. By running the executable, you many find the answers to your questions.

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

2. Design:

For this assignment, you have five 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. Third, you must store each course so that actions specified by the menu choices may be carried out. Fourth, you must decide the maximum number of courses can be stored (we will use **10** for now). Finally, you must implement the GPA calculation for all courses, the printing of all courses, calculating the number of credit hours with a grade of 'D', the GPA calculation for a particular semester, and the addition of a new course to the course list. 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.

Once the menu option 'D' or 'd' is picked by the user, your program should ask the user to enter a season and a year (e.g. Spring 2015).

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) the semester the course was taken (for example, Fall 2014), 4) course number (for example, csce2004), 5) course grade (for example A), 6) course credit hours (for example 4), and repeat 2) to 6) for the rest of the courses. The following is an example of what the user types (program output is not shown):

```
// What the user types to enter the course information, similar to hw2
2
Programming Foundations 1
Spring 2014
csce2004
A
4
Calculus 1
Fall 2013
math2554
B
4

// What the user types to enter menu options, for listing,
// gpa calculation, and quitting
B
a
Q
```

The course input format is used for course listings.

3. Implementation:

We will use parallel arrays to store course information, where each array is used to store one piece of the information associated with a course, such as a course grade array, a course hour array, and so on. 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. Make sure to make a copy of your original code. If something goes wrong, you will be able to compare with the previous version of your program.

You may begin with the hw2 code without adding menu handling code at all. In the hw2 code, use arrays and replace variables with array names and indices to read in and store course information. Once course information is stored in the arrays, you should compute GPA as in hw2 and list all the course information once. Now you have the data in the arrays and should work with the menu part of the project. (This is merely a suggestion, though.)

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 not counting program testing output) 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 could we simplify the design of our program or code so it is more modular?

How could we store all the courses (represented by the 5 pieces of data) to a file and read in the information from a file?

In addition to the 5 pieces of data associated with a course, is there other information or any other attributes we could add to a course?