CSCI 2720
Assignment 1: Introduction to C++ Programming
Due: Wednesday Sep 08 11:59 PM

Introduction to the Program

In this assignment, you will write a program to warm up your C++ skills. This in turn will help you to prepare well for the upcoming assignments.

You will create a program to store and view grades of a 'Data Structures' class. This class has 20 students and 3 instructors. Instructor details, student details and grades are stored in two different plain text files namely instructors.txt, students.txt. Contents of these two files are described below.

1. instructors.txt
    - Each line of this file represents an instructor.
    - Each line has a username, password and full name separated by tabs.
    - Username is unique for each instructor.
    - This file is provided to you.
    - The provided file has 3 lines for 3 instructors.

2. students.txt
    - Each line of this file represents a student.
    - Each line has username, password, full name, project grade, quiz grade, midterm grade and final exam grade separated by tabs.
    - username is unique for each student.
    - Grades will only be positive integers.
    - This file is provided to you.
    - The provided file has 20 lines for 20 students.

Your program allows the users to login as a student or as an instructor. Depending on the role the user logged in can perform different tasks.

If the user logged in as a specific student, the user can view his/her grades for project, quiz, midterm, and final exam. The user also can view the total course grade. Overall course grade should be calculated using the following weights.

- project - 30%
- quiz - 10%
- midterm - 20%
- final exam - 40%

If the user logged in as a specific instructor, the instructor can perform the following tasks.

- View grades of a specific student by entering the student's username.

- View the min, max and average grades of all students for project, quiz, midterm or final exam separately.
- View the min, max and average of total course grade of all students.

Structure of the program and user interactions

You should create the following two classes and main.cpp to implement this program.

1. Student
   - Student class should have the following methods with the exact prototype.
     - bool login(string username, string password);
     - string getStudentName();
     - int getProjectGrade();
     - int getQuizGrade();
     - int getMidtermGrade();
     - int getFinalGrade();
     - double getOverallGrade();
   - You should add necessary member variables including the following,
     - string fullName;
     - int projectGrade;
     - int quizGrade;
     - int midtermGrade;
     - int finalGrade;
   - Above functions must be present in the class.
   - You can add additional functions if you wish such as the following.
     - void setStudentName(string fullName);
     - void setProjectGrade(int grade);
     - void setQuizGrade(int grade);
     - void setMidtermGrade(int grade);
     - void setFinalGrade(int grade);

2. Instructor
   - Instructor class should have the following methods with the exact prototype.
     - bool login(string username, string password);
     - string getInstructorName();
     - Student getStudent(string username);
     - Student getMinStudent(int gradeType);
     - Student getMaxStudent(int gradeType);
     - double getAvg(int gradeType);
   - Note: gradeType variable in the above methods should be recognized from the following integers.
     - project: 1
     - quiz: 2

        - midterm: 3
        - final: 4
        - overall: 5
    - You should add necessary member variables that includes the following:
        - string fullName;
    - You can add additional functions if you wish such as.
        - void setInstructorName(string fullName);

****Note: As you can see from the description above you are allowed to add extra methods and extra data members so feel free to add any method or data members to your classes to correctly implement the logic of the assignment (explained below in the document). However, methods and data members given in the class description above should be present in your implementation with the same prototypes**

Main method of your program should ask the following exact question from the user.

```
User types,
     1 - Instructor
     2 - Student
Select a login user type or enter 3 to exit:
```

The user should be able to enter 1 or 2 depending on who he/she likes to login as. If the user enters anything other than 1, 2 or 3 the following message should be shown and the above question should be asked again.

```
Invalid option. Please enter a valid option.
```

If 3 is entered, the program should exit.

Login Flow

If the user selects to login as an instructor the main method should ask the following question.

```
Enter credentials to login,
     Enter username:
```

When the user enters the username the password should be asked as shown below.

```
     Enter password:
```

Then the main method should initialize an Instructor object and call its login method with the user name and the password that was read from the input. The login method should read the instructors.txt file and check if the username and the password matches any of the instructors in the text file. If they match, the full name of the instructor should be stored in the instructor object. In this case, the login method should return true. Show the following message from the main. Here the name of the instructor is assumed to be ‘John Smith’.

```
You are now logged in as instructor John Smith.
```

If the username or password did not match, show the following message from the main. Use the return of the login method to decide the message.

```
Login as instructor failed.
```

You should do the same if the user selects to login as a student. Here you should initialize a student object and call its login method in the same way as explained above. In that case you should print the exact same messages for student login flow just by replacing the word instructor with the word student. If the user successfully logs in as a student you should also store the grades of the logged in student in the student object.

Student Interactions

If the user has successfully logged in as a student, the following question should be asked.

```
Do you want to view grades (y/n)?
```

The user should be able to enter ‘y’ or ‘n’ for the above question. If ‘n’ or anything other than ‘y’ is entered the login options should be asked again. If the user entered ‘y’, the grades should be displayed as shown below.

```
Student name: John Smith
     Project    80%
     Quiz       95%
     Midterm    96%
     Final      98%
     Overall    91.9%
```

After showing the results, the program should ask login options again.

Instructor Interactions

If the user has successfully logged in as an instructor, the following question should be asked after the login flow.

```
Query options,
     1 - view grades of a student
     2 - view stats
Enter option number:
```

The user should be able to enter 1 or 2 for the above question. If the user enters anything other than 1 or 2 the following message should be shown and the above question should be asked again.

```
Invalid option. Please enter a valid option.
```

View Grades by username

If the user selects option 1. The following question should be asked.

```
Enter student username to view grades:
```

The user should be able to enter the username as a string. After the user enters a student's username the program should call the getStudent() method of the instructor object to receive a student object for the specific student. If the student is not found, the program should print the following message and ask query options again.

```
Student username is not valid.
```

If the student is found, show the grades of the students in the following format. Here the name of the student is assumed to be 'John Smith' and grades are example values.

```
Student name: John Smith
     Project    80%
     Quiz       95%
     Midterm    96%
     Final      98%
     Overall    91.9%
```

After showing the results, the program should ask to login again by selecting user types.

View Stats

If the user selects option 2 to view the stats, the program should ask the following question.

```
Grade types,
         1 - Project grade
         2 - Quiz grade
         3 - Midterm grade
         4 - Final grade
         5 - Overall grade
Select a grade type to view stats:
```

The user should be able to enter 1, 2, 3, 4 or 5 for the above question. If the user enters anything else, the following message should be displayed and the program should ask the question again.

```
Invalid option. Please enter a valid option.
```

If the user enters a valid option, the following information should be displayed. Appropriate functions from the instructor object should be called to display the following information. Assume that the user selected option 5 in the following example.

```
Overall grade stats,
     min  78% (John Smith)
     max  98% (Jack Smith)
     avg  88.5%
```

After showing the results, the program should ask to login again by selecting user types.

Data Storage Options

Below are some options you might wish to consider in order to store student data, so as to make instructor stat methods easier to implement. You are **NOT** required to use any of these approaches. They are here purely as reference.

Static Array

One approach is to take advantage of the class structure in C++ and add a static array of student objects to the instructor class definition in Instructor.h using one of the two following lines of code:

```
static Student arr[20]; // to implement as an array with fixed size.
static Student * arr; // to implement as a dynamic array. Note: You
have to use the 'new' keyword to initialize the array later eg: new
Student[20]. And also you should delete the array when you no longer
need it to avoid memory leaks.
```

Note that simply adding a line above and trying to access the array outside of the header file will yield undefined reference errors. In order to make them accessible, somewhere at the top of your Instructor.cpp file (below the includes but above the function implementations), add one of the following, depending on whether your array is fixed or dynamic:

```
Student Instructor::arr[] = {}; // for fixed-size array
Student * Instructor::arr = {}; // to implement as a dynamic array.
```

You may then populate the array with appropriate Student objects either in main.cpp or through a helper method in Instructor.cpp.

Local Array

Another approach is to create an array in main.cpp, initializing it using one of the two following lines of code:

```
Student arr[100]; // to implement array with fixed size
Student * arr; // to implement as a dynamic array. Note: You have to use the 'new' keyword to initialize the array later eg: new Student[20]. And also you should delete the array when you no longer need it to avoid memory leaks.
```

You may then populate the array with appropriate Student objects as you see fit.

Vectors

One last approach is to use vectors, the C++ equivalent of Java ArrayLists. You will need to have an include for them:

```
#include <vector>
```

To initialize an empty vector of students, add the following line of code:

```
std::vector<Student> v;
```

Like with ArrayLists in Java, the data type specified inside the angle brackets denotes the type of objects that will be stored in the vector. Vectors have variable size and have a few helpful functions, which you may explore and use as needed. To add a Student object to the vector, use the push_back() function as such:

```
Student s1; // empty Student object
v.push_back(s1) // adds s to the end of the list
```

Note that you may also use square brackets to access/modify elements in a vector, but they can only be used on indexes that already have an object assigned to them. This means that they can be used to replace elements, but cannot be used to add elements beyond the end of the vector, so either use push_back() for that purpose or specify a starting size for the vector using one of its constructors.

```
Student s2 = v[0]; // OK - accesses the empty Student object we added
Student s3; // empty Student object
v[0] = s3; // OK - assigns s3 to index 0, replacing s1
v[1] = s3; // ERROR - results in segmentation fault. Use
// v.push_back(s3) instead
```

You may combine the use of vectors with any of the two approaches described above (or another approach of your choice), keeping the code provided the same but replacing “Student”

with “vector<Student>”. Vectors eliminate the need to allocate memory dynamically when having a list of variable size, and provide several methods/utilities that are not available with normal arrays. Be sure to take them into consideration when deciding how to store your data.

**** Note: Below are sample outputs for this assignment that will show how you need to display the outputs for different cases. You should use the exact same messages and formatting as shown in the sample output below in your own implementation. Copy them from the instructions document (sample output below) directly to your code to avoid mistakes. The sample output also shows you the amount of error check that you need to have in your program. You don’t need to worry about any other types of error check as long as your code handles the error check cases below your are all good. In short you just need to make sure that your code is able to replicate the sample output below. Make sure to test your code as shown below in the sample output.**

Read instructors and students filenames from command line

You should not hard-code the information of students and instructors. Instead, your program should read information from the textfiles. The filenames containing the instructors and students' information should be read from the command line. (Recall that you can use argc and argv to retrieve command line arguments.)
As a correct example, your program should run with the following command syntax:

```
Usage: main [instructers_file] [students_file]
Example: $ ./main instructors.txt students.txt
Parsing instructors and students information success.
...
```

If the command file arguments is not equal to two, your program need to prompt the correct error message and quit:

```
$ ./main instructors.txt
Usage: main [instructors_file] [students_file]
```

If the command file arguments given to the program are not correct file names, or the files cannot be processed successfully, your program should give a friendly read message and quit:

```
$ ./main not_a_file_name students.txt
Error: cannot parse instructors information from file not_a_file_name

$ ./main instructors.txt not_a_file_name
Error: cannot parse students information from file not_a_file_name
```

Sample Output

Below are three different sets of sample outputs that show the flow and output of the program.

Sample 1 shows the proper prompts and responses for when you try to login as an instructor. It shows how to handle some invalid input and what to print when you choose to view grades for a particular student or the stats for the entire class.

Sample 2 shows the proper prompts and responses for when you try to log in as a student. It shows how to show a user their grades.

Sample 3 shows the various responses when invalid input is given when the login prompt is shown.

Note: Your code should use the exact same prompts and look exactly like the sample code below.

Sample 1 (Instructor):

```
User types,
        1 - Instructor
        2 - Student
Select a login user type or enter 3 to exit: 1

Enter credentials to login,
        Enter username: u000537
        Enter password: pw9836

You are now logged in as instructor Luke Palmer.

Query options,
        1 - view grades of a student
        2 - view stats
Enter option number: 3

Invalid option. Please enter a valid option.

Query options,
        1 - view grades of a student
        2 - view stats
Enter option number: 1

Enter student username to view grades: user1

Student username is not valid

Enter student username to view grades: u000534
```

```
Student name: Billy Robertson
        Project 84%
        Quiz    89%
        Midterm 94%
        Final   83%
        Overall 86.1%

User types,
        1 - Instructor
        2 - Student
Select a login user type or enter 3 to exit: 1

Enter credentials to login,
        Enter username: u000537
        Enter password: pw9836

You are now logged in as instructor Luke Palmer.

Query options,
        1 - view grades of a student
        2 - view stats
Enter option number: 2

Grade types,
        1 - Project grade
        2 - Quiz grade
        3 - Midterm grade
        4 - Final grade
        5 - Overall grade
Select a grade type to view stats: 6

Invalid option. Please enter a valid option.

Grade types,
        1 - Project grade
        2 - Quiz grade
        3 - Midterm grade
        4 - Final grade
        5 - Overall grade
Select a grade type to view stats: 3

Midterm grade stats,
        min     78% (Cody Gilmore)
        max     98% (Max Whitley)
```

```
        avg     88.7%

User types,
        1 - Instructor
        2 - Student
Select a login user type or enter 3 to exit: 3
```

Sample 2 (Student):

```
User types,
        1 - Instructor
        2 - Student
Select a login user type or enter 3 to exit: 2

Enter credentials to login,
        Enter username: u000534
        Enter password: pw4096

You are now logged in as student Billy Robertson.

Do you want to view grades (y/n)? y

Student name: Billy Robertson
        Project 84%
        Quiz    89%
        Midterm 94%
        Final   83%
        Overall 86.1%

User types,
        1 - Instructor
        2 - Student
Select a login user type or enter 3 to exit: 2

Enter credentials to login,
        Enter username: u000534
        Enter password: pw4096

You are now logged in as student Billy Robertson.

Do you want to view grades (y/n)? n

User types,
        1 - Instructor
        2 - Student
```

```
Select a login user type or enter 3 to exit: 3
```

Sample 3 (Invalid Input):

```
Select a login user type or enter 3 to exit: 4

Invalid option. Please enter a valid option.

User types,
        1 - Instructor
        2 - Student
Select a login user type or enter 3 to exit: 1

Enter credentials to login,
        Enter username: user1
        Enter password: pass1

Login as instructor failed.

User types,
        1 - Instructor
        2 - Student
Select a login user type or enter 3 to exit: 2

Enter credentials to login,
        Enter username: user1
        Enter password: pass1

Login as student failed.

User types,
        1 - Instructor
        2 - Student
Select a login user type or enter 3 to exit: 3
```

Grading Rubric

Implementation	Grade
Readme, comments, and makefile	9.00%
Student login	15.00%
Instructor login	15.00%

Student grade display	10.00%
Instructor check student grade	15.00%
Instructor view min	12.00%
Instructor view max	12.00%
Instructor view avg	12.00%
Total	100.00%

Compiling the Program (Same as previous assignments)

A Makefile should compile your code into a program called “main”. Your program should run with the following command syntax:

```
Usage: main [instructor_file] [student_file]

$ ./main instructors.txt students.txt
```

Commands to run and compile the code should be clearly documented in the Readme file. **Code that fails to compile or the code that compiles but fails to run will receive a grade of zero.**

Late Submission Policy

Except in the cases of serious illness or emergencies, projects must be submitted before the specified deadline in order to receive full credit. Projects submitted late will be subject to the following penalties:

- If submitted 0–24 hours after the deadline 20% will be deducted from the project score
- If submitted 24–48 hours after the deadline 40% points will be deducted from the project score.
- If submitted more than 48 hours after the deadline a score of 0 will be given for the project.

Students unable to submit a project due to a serious illness or other emergency should contact the instructor as soon as possible before a project’s deadline. Based upon the circumstances, the instructor will decide an appropriate course of action.

Submission Notes

You must include your full name and UGA email address in your Readme.txt file. If you are doing a project in a group of two, list the full names and UGA email addresses of all two group members. If you are in a group you must also describe the contributions of each group member in the assignment in the readme file.

You must submit all the following files on Odin:

- **Makefile**
- **main.cpp**
- **Student.h**
- **Student.cpp**
- **Instructor.h**
- **Instructor.cpp**
- **students.txt**
- **instructors.txt**
- **Readme file with all necessary descriptions.**

Odin Login Instructions

This semester we are switching from Nike to a new server called Odin. Unlike with Nike, to use Odin, you must use the UGA VPN first. Please visit remote.uga.edu if you haven't already and install the VPN client for your OS. Be sure to have an SSH client like PuTTY or MobaXterm and use that to connect. The server's name is odin.cs.uga.edu and you will log in with your MyID credentials. For those that have used Nike before, the login process is almost the same, except that nike.cs.uga.edu is replaced by odin.cs.uga.edu, and, as stated before, you'll log in with your MyID credentials instead of your regular Nike credentials.

Group Submissions:

****Note: Assignment can be done individually or in a group of two students. The members of the group should be from the same section. Submit a readme file with your names, UGA emails and any other information that you want to give the TA to run your code. If you are doing it in a group of two make sure to include the contribution of each student in the readme file.**

If you are working in a group of two, your submission folder must be named as below:

```
LastName1_LastName2_assignment1
```

When you are submitting the folder you should use the submit command as below:

```
$ submit LastName1_LastName2_assignment1 csci-2720b
```

If you are working in a group, you must **submit the assignment from the Odin account of only one of the members** in the group.

Individual Submissions:

If you are working individually, your submission folder must be named as below:

```
LastName_assignment1
```

Submission command should be used as below:

```
$ submit LastName_assignment1 csci-2720b
```

****For all the submissions, the folder naming and the readme is very important in order to be able to locate your assignment and grade it.**

****Note: Make sure you submit your assignment to the right folder to avoid grading delays.**

When using the submit command, first set your current directory of the terminal to the immediate parent of the assignment folder.

You can check your current working directory by typing:

```
$ pwd
```

For example if your assignment is in the following directory,

```
/documents/csci2720/assignments/LastName_assignment1
```

Before executing the submit command, change to the assignment's immediate parent directory:

```
$ cd /documents/csci2720/assignments/
```

Then execute the submit command:

```
$ submit LastName_assignment1 csci-2720b
```

To check if your submission command was valid, there should be a receipt file.

```
$ ls LastName_assignment1
// should return rec##### like 'rec54321'
```

Selecting the Odin Destination

You must select your Odin submission destination using the following mapping based on the section of the course you are registered to.

Section	Class Time	Odin Submission Account
CRN 17504	M 10:20 AM	csci-2720a
CRN 42634	M 04:10 PM	csci-2720c