CSE 1384 - Classes
Lab 5

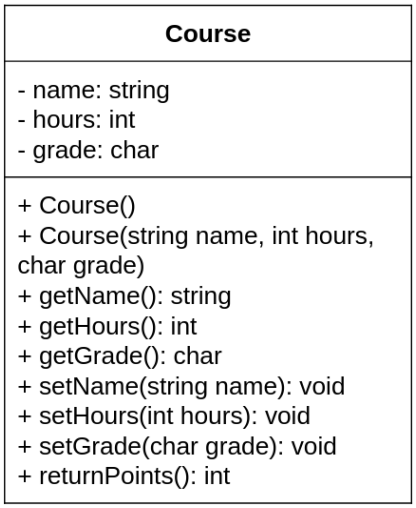
Objectives:

- Continue practicing past concepts
- Practice using basic C++ classes

Assignment:

Part one:

Build a class that's defined by the detailed UML class diagram found below. For a UML cheat sheet, please consult the last page of the lab details. It breaks down the diagram structure there.



You should have basic getters/setters for all three data variables, two constructors (one an empty constructor, one that takes in all three variables as parameters), and a final function returnPoints that should return: 4 for the grade being an 'A', 3 for the grade being a 'B', 2 for the grade being a 'C', 1 for the grade being a 'D', and 0 for the grade being an 'F'.

This should result in two files: `course.cpp` and `course.h`. Test that your class is made correctly by using the included `testClass.cpp`. This should work with ZERO changes with your class.

Part two:

Now, you must complete the `main.cpp` provided to you that is using your class. The overall goal of this program is to take a given schedule and provide the GPA to the user using quality points (which is what our university does). Portions of your main and some functions have already been provided to you. These include:

- A display function for your schedule once it's read in from the file
- Three functions overall that will create the GPA output and that you can use to help in items in the program
- Some main function items:
    - File reading verification (three sample schedules have been provided to you in order for you to use)
    - The start of the menuing

You must finish/provide:

Reading from your file. Notably, the file contents are stored into a Course vector (using the class you built!). The file you plug in should follow the following format:

```
Name of the class
Hours Grade
```

Read in each class and add each one to your vector.

Finishing the menu: add and remove options in the menu are not complete. Add them! Both menu options should call their respective functions. They also both should have some error checking before doing so: checking if the user is at or above 19 hours (adding) or if the schedule is empty (removing).

Finally, you must finish the functions in the program.

- addClass
    - Add a class to the vector!
    - Error checking should include:
        - Hours can only be 1-4
        - Grades must be: A, B, C, D, F (caps matter!)
- removeClass
    - Remove a class from the vector!
    - Display the names and a number starting at 1 to help your user out
    - Error checking should include:
        - Entering in a correct number given the range

Milestones:

You must receive a check-off for each item to receive full credit on the lab. You may show the milestones incrementally, or receive a check off for multiple in one demo depending on your progress.

Examples of each milestone are included.

Milestone 1:

Class creation. Must show:

- Your class working with `testClass.cpp`
    - Output should be identical
- This milestone MUST be shown – you cannot just finish the lab and only show the last milestone. At bare minimum, milestone 1 and the last milestone must both be shown separately to ensure the class fits the UML design

```
Course object 1:
Name: Test Class 1
Hours: 3        Grade: A
Resultant grade points: 4

Course object 2:
Name: Test Class 2
Hours: 4        Grade: C
Resultant grade points: 2
```

Milestone 2:

File reading into the vector and class compatibility. Must show:

- The display menu provided option working
- The calculate GPA menu option working

(Example of the first three classes being displayed from schedule.txt – abbreviated for space purposes)

```
Course: Intermediate Computer Programming
Hours: 4        Grade: B

Course: Calculus II
Hours: 3        Grade: C

Course: General Psychology
Hours: 3        Grade: B
```

Milestone 3:

Completed menu options and error checking added to them. Must show:

- Error messages on both
    - `19hours.txt` can be used for the add menu option
    - `empty.txt` can be used for the remove menu option
- A call to the appropriate function otherwise
    - You can use the opposite file for each (`empty.txt` for add, for example) or `schedule.txt` for both

Errors:

```
Enter a file name: empty.txt

File reading...
Reading complete. File closed.

Welcome to the GPA manager.

0. Exit
1. View Schedule
2. Add Class
3. Remove Class
4. Display GPA
Enter menu option: 3

You have no classes in your schedule. Please try adding them first.
```

```
Enter a file name: 19hours.txt

File reading...
Reading complete. File closed.

Welcome to the GPA manager.

0. Exit
1. View Schedule
2. Add Class
3. Remove Class
4. Display GPA
Enter menu option: 2

You can't add more once you're at 19 hours! Try removing a class first.
```

Valid:

```
Enter a file name: schedule.txt

File reading...
Reading complete. File closed.

Welcome to the GPA manager.

0. Exit
1. View Schedule
2. Add Class
3. Remove Class
4. Display GPA
Enter menu option: 2

Add Class function called!
```

Milestone 4:

Completing the add class function. Must show:

- All appropriate error messages looping
- A new class added to the schedule following a display

```
Enter menu option: 2

Enter the course name: This is a class
Enter the number of hours the course is: 0

Invalid hour amount. Please try again.
Enter the number of hours the course is (1-4): 3
Enter your course grade: a

Invalid grade. Please try again.
Enter your course grade (A, B, C, D, F): A
```

Milestone 5:

Completing the remove class function. Must show:

- All appropriate error messages looping
- A class removed from the schedule following a display

```
Enter menu option: 3

1: Intermediate Computer Programming
2: Calculus II
3: General Psychology
4: Discrete Structures
5: Chemistry I
6: Chemistry Lab
7: This is a class

Which class would you like to remove: 0

Invalid class. Please try again.
Which class would you like to remove: 8

Invalid class. Please try again.
Which class would you like to remove: 7
```

Comment Block:

Your code should contain a comment block at the top containing information on who wrote the code, what the assignment is, when it is due, etc. Here is an example of a good comment block to put:

```
/*
 Name: <your name>                          NetID: <your netID>
 Date: <current date>                       Due Date: <enter in due date>

 Description: <What is the program?>
*/
```

Deliverables:

- Code for the Course class (.cpp and .h file)
- C++ code for the main file (.cpp file) you used to demo your program
    - If the code submit CANNOT achieve the demo, you will receive a 0.

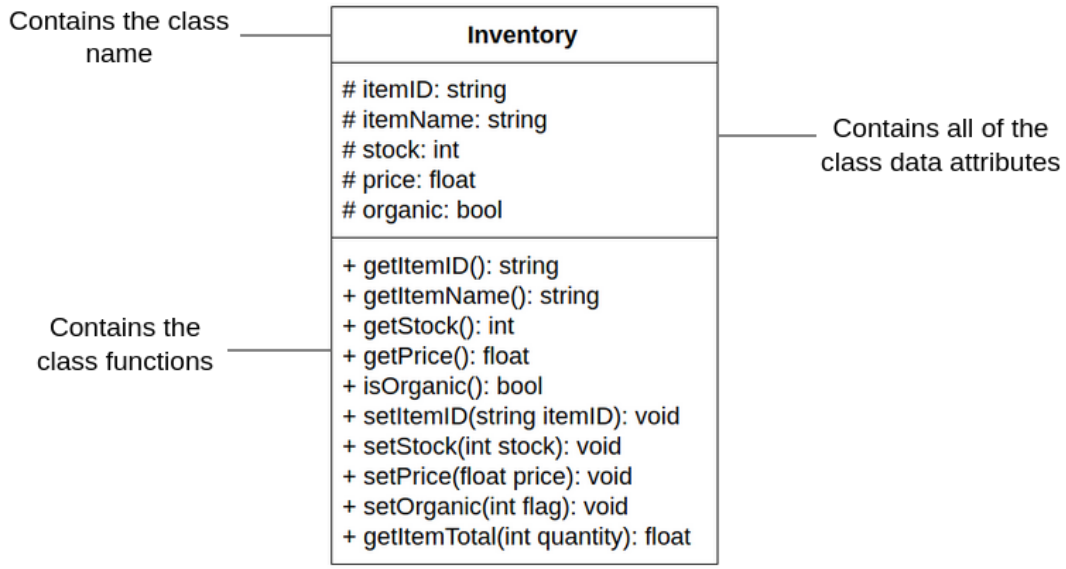
Point Breakdown:
(100 points total)
A submission that hasn't received a check-off will not be considered for grading.

- 25pts - milestone 1
- 15pts - milestone 2
- 10pts - milestone 3
- 20pts - milestone 4
- 20pts - milestone 5
- 10pts - programming style *
- (PENALTY) 15pts - changing the code provided to you

* Programming style includes good commenting, variable nomenclature, good whitespace, etc.

UML Cheat Sheet:

Each portion of a detailed UML Class diagram contains a different part of the class. There's a section for the name, variables, and functions belonging to the class. A breakdown of these is found below.



Additionally, each variable and function will be preceded by a symbol: +, -, or #. These refer to where your data will be accessible. Each of those mean the following:

- - Private
- + Public
- # Protected

For your variables, each will be followed by the data type they should be initialized as. So, "# itemID: string" should be a *protected string variable named itemID.*

Finally, your functions contain any/all parameters (and their data types!) following the function name. They then are followed by a colon and the data type they should return. So, "+ getStock(): int" should be a *public function that has no parameters and returns an integer*.