**NAME**_____

**CSIS 137 – Advanced C/C++**
**Midterm Project – Fall 2017**
**Due 10/29/17 (by 11:55 pm) – Submit on Canvas by the Due Date (NO LATES)**

For this class you will be doing a midterm project – due to time constraints I only have one option for you to choose from for this project. **This DOES NOT MEAN any TWO PROJECTS SHOULD LOOK EXACTLY ALIKE (if your program looks too much like someone else's I will ask you to explain your code, if you can't we'll go to the next step).** The project leaves a lot of things open and EVERY PROJECT should be different. This project incorporates some of everything we've done so far in this class (class basics, composition, cascading, operator overloading etc.)

***GROUP OF TWO IS REQUIRED:*** For this project I'm requiring groups of two (I will allow the group size to go UP TO 3, but no more than 3) – you may choose your group. **So that means a minimum of two in a group and a maximum of three in a group. The week of Sept 27th our online activity will just be forming groups – the easy online activity will give you time to start the project and not procrastinate.** I decided to require the groups for two reasons:

- To meet one of the Business Division learning objectives: that students learn team work.
- To make it so I can grade papers faster and give better feedback.

You will NOT be working in a group for the final and I never require groups for homework – so if you dislike groups this is the ONLY time you will have to work in one.

***IMPORTANT:*** ***Make sure and follow the design principles we have used in class***, I didn't specify when to use const, how to set up constructors, etc. etc. - you should know this and use the best software engineering method based on what we learned in class. Use the example programs as guides to help you. It is NOT ok for it to "just work", it needs to work efficiently and employ the best software engineering principles based on what we have learned.

You may add additional features to the project if you find it too easy – BUT – make sure and meet all of the requirements first. I will add points if you need them for implementing extra features but there will not be any extra credit. In other words the extra points can only bring your score to a maximum of 100%.

**NO LATES ON THIS MIDTERM PROJECT…make it a priority – DO NOT PROCRASTINATE!!!!!!!!**
***If you don't get it all working submit what you have for partial credit! Don't submit nothing – nothing means a zero, handing in something means partial credit.***

## MIDTERM PROJECT – Class Schedule and Schedule Info Project

For this project you are going to create several classes that will work together in a project to let a student/user enter courses and display a class schedule; the project will let the user both remove and add courses to their schedule as well as calculate some information such as the duration of each daily class. It won't be a perfect project (we haven't gotten far enough in the book for that yet) but you will also implement some checks and restrictions on the various data. Make sure and implement each class below. I have given some hints to help you avoid pitfalls I ran across when implementing this. Make sure and read ALL of the instructions first.

Not that anyone in this class would do this but this project is NOT the same as last semester, it has the same idea but there are additions/deletions and changes so make sure to complete THIS project. Completing last year's project does not get you any points – complete THIS year's project.

### Step 1: Modify the `Time` Class

Take the **`Time`** class from Chapter 10 – Example 10 (if you don't have the Example I will post the .cpp and .h files Midterm project folder as well). Modify the **Time** class in the following ways so it will work better with a Class Schedule.

1. Remove of the "seconds" member variable. You will also need to remove the corresponding member functions and any functions where "seconds" was used as a parameter. NOTE: Internally Time should still be kept as universal – so you should not add any new member variables to the class.
2. Create a utility function that converts standard time to universal time. This will help you in some of the remaining calculations. Implementation is up to you but try to use the best design possible.
3. Overload the **`<<`** operator for this class
   - have it output the time similar to the way the **`printStandard`** function does but take out the seconds and the **`endl`**
   - Make sure cascading is allowed
4. Overload the **`>>`** operator for this class.
   - The operator should allow the user to enter the input in STANDARD time (which the user would be more familiar with) NOT in universal time. For example: the user should enter 6:20 PM not 18:20.
   - Have the **`>>`** operator accept input in this form: **`HH:MM AM (or PM)`**. So first the hour is entered followed by a colon followed by the two digit minute followed by a space and then AM or PM. If the hours do not need two digits you should be able to have the user enter 6 rather than 06 – account for this in your function. Set up some defaults for cases where the user enters inappropriate data for hours, minutes or AM/PM. Nothing fancy since we don't know exception handling but don't use bad design (for example using cout to say "error" in the member function or handling it procedurally).
   - Keep in mind even though the user is entering the data in Standard time format internally time is universal so the function should handle this appropriately
   - Cascading should be allowed for this operator
5. Overload the **`-`** (subtraction) operator. This function will take two **`Time`** instances and calculate the hours and minutes between them. This means the function should return the hours and minutes between the two **`Time`** instances. It should return the hours and minutes as a decimal. So if the two times are 10:30AM and 10:45PM the function will return 12.25 hours. To get the fractional part you just divide the minutes by 60 (this means 15 minutes is .25 hours (15/60)). Note you will need to come up with a formula for calculating the difference between two times. I found one quickly with Google.

### Step 2: Modify the `Date` Class

Take the **`Date`** class from Chapter 10 – Example 4 (if you don't have the Example I will post the .h and .cpp files the Midterm project folder as well). Modify the **`Date`** class in the following ways so it will work better with a Class Schedule.

1. Remove the **print** function and all occurrences of it. Also remove any printing done in the constructor and remove the destructor.
2. Add get and set functions for all the member variables. Make the set functions allow for cascading (see Chapter 10 if you forgot what this is)
3. Overload the **<<** operator for this class
    - ♦ Have the it output the time similar to the way the **print** function did but take out the **endl**
    - ♦ Cascading should be allowed for this operator
4. Overload the **>>** operator for this class
    - ▪ Have the **>>** operator accept input in this form: **MM/DD/YYYY**. So first the two digit month is entered followed by a dash, followed by the two digit day, followed by a dash, followed by the four digit year. If the month and/or the day are entered in single digit form by the user (ie 6/6/2017) the function should still perform properly as if 06/06/2017 was entered.
    - ▪ Cascading should be allowed for this operator
5. Overload the **>**, **<**. **>=**, **<=**, **==**, and **!=** operators for this class. Two **Date** instances are equal if their hours and minutes are the same. To determine if one **Date** instance is greater than another first check to see if the hours are greater than the other, if so the one with the bigger number in hours is bigger. If the hours are equal then check the minutes variable, the **Time** instance with the bigger minutes variable is the bigger one.
    - ♦ Be efficient – reuse code where you can

## Step 3: Create a Course Class (both the .h and the .cpp separate like the others)

A course is a data type with information in it about just ***ONE*** course.

Create a class **Course** with the following member variables:

- ▪ a course number (example "CSIS 112")
- ▪ a course name (example: "Java")
- ▪ a variable for the course meeting days (example: "TTH") – I realize this is not good design here, but our project already has enough in it
- ▪ a variable representing the number of units the course is worth (example 4.0 units)
- ▪ a **Date** object for the start date of the class
- ▪ a **Date** object for the end date of the class
- ▪ a **Time** object for the start time of the class
- ▪ a **Time** object for the end time of the class.

IMPORTANT: Do not make any of these member variables **const** unless you want the added task of overloading the assignment operator later.

The **Course** class should have the following member functions

1. Create a constructor that:
    - o Takes eight arguments and uses default values for each argument (you decide the default values)
        - ▪ HINT: for object type member variables you can put calls to their default constructors as default values (Example: ConstructorName(ObjectName = ObjectName()); )
2. Create a destructor that prints a message saying a course has been deleted ( I know this is not practical but this is to show me you understand the destructor).
3. Create get functions for the following member variables: the course number, the course name, the course meeting days, and the number of units the course is worth.
4. Create get functions for the four Date and Time member variables BUT you must NOT rewrite code, you must reuse code from the Date and Time classes. Do not recreate "Time" or "Date" code in the Course class.
5. Create set functions that ALLOW CASCADING for the following member variables: the course number, the course name, the course meeting days, and the number of units the course is worth.

6. Create set functions for the four Date and Time member variables BUT you must NOT rewrite code, you must reuse code from the Date and Time classes. Do not recreate "Time" or "Date" code in the Course class.
7. Overload the **<<** operator for this class. Have your course display in a way similar to what is shown below.

```
Course Info:    CSIS 112 -- Java
# of Units:     3.00
Course Dates:   09/06/2017 - 12/18/2017
Meeting Days:   T
Meeting Time:   6:55PM - 10:05PM
Daily Duration: 1.42 hours
```

   Make sure and USE the overloaded **<<** you created for the **Time** class and **Date** class above in this function. Do not repeat code already in the **Time** or **Date** classes.

8. Create a function named **calcDailyDuration** which calculates the hours the student will be in the class for one daily class meeting. The function should calculate fractionally. So if the class is 1 hour and 30 minutes long the daily duration is 1.5 hours. (Note: don't repeat code here – you already have code that calculates this somewhere)

## Step 4: Create a Semester class (both the .h and the .cpp separate like the others)

A Semester is a data type with information in it about one school semester.

Create a class **Semester** with the following three member variables:

- a semester name (Example: Fall 2017)
- a Date instance for the start date of the semester
- a Date instance for the end date of the semester

The **Semester** class should have the following member functions

1. Create a constructor that: accepts three arguments, the semester name, the start Date and the end Date. Use default values for all the parameters in the constructor
2. Overload the << operator for this class
   - Have it output the semester name and dates in a manner similar to this:
     **Semester: Fall 2017 (09/03/2017-12/12/2017)**
3. Overload the >> operator for this class
   - Have the >> operator accept input for the Semester name, start date and end date. You can choose the best formatting.
4. Create get and set functions for each member variable – DO NOT recreate the Date class here – reuse code as appropriate.

## Step 5: Create a CourseSchedule Class (both the .h and the .cpp separate like the others)

A **CourseSchedule** is a data type that is a list (array) of **Course** data types. You will need to have a pointer to a **Course** data type so our list (array) can be of any size. This is similar to what you did in Homework #2 except for your array is of type **Course** instead of type **int**.

Create a class **CourseSchedule** with the following member variables:

- A student name (the name of the student this schedule is for)
- A **Semester** object (see above)
- A pointer to a **Course**

  **This will allow you to dynamically allocate memory so the class will work for any size student schedule (the student can have as many classes as they want on their schedule)

- a **maxSize** variable that indicates the maximum number of courses that can be added to this student schedule (the array of **Courses**)
- a **numCourses** variable that keeps track of how many courses have been added to the student schedule (the array of **Courses**)

The **CourseSchedule** class should have the following member functions:

1. Create a constructor that:
    - Takes three arguments and sets the following member variables appropriately: the student name, the Semester, and the maximum number of courses the student is allowed to take.
    - The constructor should set **numCourses** equal to zero since at this point no specific courses have been added to the student's schedule
    - The constructor should dynamically allocate the array of courses to be "maxSize" big (so dynamically allocate an array of **maxSize Course** instances)
2. Create a destructor that de-allocates memory appropriately when a **CourseSchedule** is destroyed. If you have any memory issues you can always put a print in here to help test.
3. Create get functions for the following member variables: the student name, the Semester (reuse code do NOT recreate the Semester data type here), and the number of courses.
4. Create set functions for the following member variables: the student name. Once the other member variables are set once the user should not be allowed to reset them (I think this could cause corruption).
5. Create a utility function named **checkDates** that takes three arguments: a Semester instance, a beginning course Date instance and an ending course Date instance. The function should check to make sure that the start date of the class and/or the end date of the class are not outside the Semester Dates. For Example: If a Semester starts 09/03/17 and ends 12/15/17 the course cannot start 08/29/17 because that is BEFORE the semester starts. The idea is a student will not be able to add a course to their schedule if that course's dates are outside the Semester beginning and end dates. (NOTE: I added this new this year, I think it should work but you may need to tweak it a bit, so this one will be flexible on number of parameters and implementation)
6. Create a function named **AddCourse** that takes a **Course** as an argument and adds this course to the **ClassSchedule** array. ***NOTE:*** the course should not be added if its dates fall outside the semester dates (see #5 above). You do NOT need to allocate any new memory since we allocated a max size for our **ClassSchedule** array. As long as you didn't make any of the **Course** member variables const you should be able to use memberwise assignment to add the course to the proper element of the array (**numCourses** is the current element) – you should add one to **numCourses** when you add a new **Course** to the **ClassSchedule**.
7. Overload the **<<** operator for this class. Have the **CourseSchedule** display in a way similar to the one shown below.

```
CLASS SCHEDULE
----------------------
Name: Jane Doe
Semester:  Fall 2017 (09/03/17-12/18/17)
Number of Classes: 2
----------------------------------------
Course Info:    CSIS 112 -- Java
# of Units:     3.00
Course Dates:   09/06/2017 - 10/15/17
Meeting Days:   MW
Meeting Time:   10:45AM - 12:10PM
Daily Duration: 1.42 hours


Course Info:    CSIS 154 -- C#
```

```
# of Units:      3.00
Course Dates:    09/06/2017 - 12/18/2017
Meeting Days:    W
Meeting Time:    6:55PM - 10:05PM
Daily Duration: 3.17 hours
```

- Make sure and USE the overloaded **<<** you created for the **Course** class (HINT: use a for loop and use the overloaded **<<** for each element of the **CourseSchedule** array since each element is a **Course**). Do not repeat code already in the **Course** class.

8. Create a function called **removeCourse** that allows the user to delete a course from their schedule. I will let you decide how to design this one. You can modify the design of the project as needed to make it happen but you don't have to – there are several ways to do this. For this one it's more about finding a solution, the best solution may be difficult since we have only done a few Chapters of the book. This is about problem solving so there is no "right" answer here so don't ask the instructor, instead problem solve on your own. (NOTE: This is a new required feature this year so I will allow leeway on it)
9. Disallow use of the copy constructor and memberwise assignment for the CourseSchedule class since it contains a pointer

## Step 6: Create a client program to test your Course and CourseSchedule classes

The program should begin by asking the user for the following information: their name, the semester name they are entering a schedule for, the dates for that semester and the maximum number of classes they are allowed to take.

The program should first create and instance of the **Semester** class by passing the information the user just entered into the constructor.

Next the program should then create an instance of the **CourseSchedule** class by passing the appropriate information into the constructor.

The program should then display the following menu (note that the semester should be the SEMESTER THE USER ENTERED, not always the same!):

```
COURSE ENTRY MENU FOR:  Fall 2017 (09/03/17-12/18/17)
------------------------------------------------------
1) Enter a new course
2) Remove a course
3) Print a Semester Schedule
q) Quit the program
```

- **If the user selects 1:** Ask the user for the course number, the course name, the meeting days for the course and the number of units the course is worth. Next use the overloaded **<<** operator from the **Time** class to ask the user the starting time for the class. Then use the overloaded << operator from the **Time** class to ask the user the ending time for the class. Finally use the overloaded **<<** operator from the **Date** class to ask the user the starting date for the class. Then use the overloaded **<<** operator from the **Date** class to ask the user the ending date for the class.

  Use the information obtained above to create an instance of the **Course** class (handle invalid courses with dates outside the semester range with the best design possible).

  Add the above instance of the **Course** class to the **ClassSchedule** instance created above using the **AddCourse** member function of the **ClassSchedule** class.

- **If the user selects 2:** Depending on how you designed the **`removeCourse`** function in your **`CourseSchedule`** class above test it here. Since I left design up to you on this function, I also have to leave this test code design up to you. I can think of a few different ways to handle this both here and in **`CourseSchedule`**. Remember this isn't graded as strictly since I left design up to you – just see what you can make happen.

- **If the user selects 3:** – Use the overloaded **`<<`** from the **`ClassSchedule`** class to print the user's schedule.

- **If the user selects 'Q' or 'q':** Quit the program

- Allow the user to continue processing strings (using the menu) until they select 'Q' or 'q' to quit

- If the user makes an invalid menu selection, print an error message to the screen