CSE30: Lab #7 – Classes

Overview

This lab will give you an insight in the power associated with Object Oriented Programming, by exploiting the concept of classes. Classes (much like structures) allow programmers to define their own "data type" by defining objects, which can be comprised of both variables and functions. This lab will re-use the code that you wrote for Lab 6, where you wrote a Time and a Course structure. In this lab, you will modify only the first part of Lab 6 so that you convert your Time structures into Time class.

In this lab you will:

- take advantage of the power of classes by using member functions, constructors and destructors, and
- implement classes and use .h and .cpp files: .h files for class declarations, and .cpp files for class definitions and the main program.

Getting started

Create a new directory in your main development directory (probably on Desktop/CSE30) called Lab_08. Try to use the terminal on your own, without getting help from the TA to setup the new directories (try to learn/remember the terminal commands).

The g++ syntax to compile classes is slightly different than for a single program comprised of a main (and potential functions):

g++ *class1.h class1.cpp class2.h class2.cpp mainSource.cpp* -o *executable*

where:

- g++ is the compiler (installed on your Linux system) of C++ source files,
- ***mainSource.cpp*** is the source file for your main program (main function),
- ***class1.h*** is the class declaration for your 1st class,
- ***class1.cpp*** is your 1st class definition,
- ***class2.h*** is the class declaration for your 2nd class,
- ***class2.cpp*** is your 2nd class definition (and so on...),
- -o tells the compiler that you want to give the executable its own name, and
- ***executable*** is the name you want to give your program.

As an example, if we have a main source file called ***main.cpp***, the class declaration called ***Time.h***, the class definition called ***Time.cpp***, and want to create an executable called ***aTestProgram***, you would type:

g++ *Time.h Time.cpp main.cpp* -o *aTestProgram*

Assuming that your program compiled successfully (i.e. no errors were found), you can run your program as you normally would by typing "***./aTestProgram***" in the terminal/console.

Good coding practices (worth 2 points!)

Writing code that is understandable by humans is as important as being correct for compilers. Writing good code will help you complete the code, debug it and … get good grades. It is very important to learn as soon as possible, because bad habits are hard to get rid of and good habits become effortless. Someone (guess who) reads your code will be in a better mood if it is easy to understand … leading to better grades! This lab will include 2 points (10% for code quality):

- Explanations with comments
- Meaningful names
- Indenting of blocks { } and nesting …
- Proper use of spaces, parentheses, etc. to
- Visible, clear logic
- One / simple statements per line
- Anything that keeps your style consistent

(Exercise 1)

This exercise is exactly the same as Exercise 1 of Lab 6. Instead of creating a Time structure you will create a Time class. The output from your program should be exactly the same as the one for Lab 6, but you will be using a class rather than a structure. In order to receive full credit for this part of the lab, you must create:

- two files for your class, a ***Time.h*** file with your class declaration and a ***Time.cpp*** file with your class definition.
- a ***main.cpp*** file for your ***main***, ***getTimeFromUser***, and ***print24Hour*** functions.
- a total of 3 ***private variables in Time class***: **hours**, **minutes**, and **seconds**.
- a total of 9 ***public functions*** in ***Time class***:
    - Default Constructor (***Time()***): initializes the three variables to 0
    - Extra Constructor (***Time(parameters)***): takes three parameters and initializes the hours, minutes, and seconds based on them.
    - Destructor (***~Time()***): does nothing.
    - 3 ***"Accessor" functions***: each one returns the current value for the hours, minutes, and seconds respectively.
    - 3 ***"Mutator" functions***: each one takes a parameter and sets it as the current value for the hours, minutes, or seconds. Inside the ***getTimeFromUser*** function, you no longer can set the hours by typing ***start_time.hours=h***

because hours is set to be ***private*** now. You have to use these mutator functions to set the time.

This is an excerpt of the relevant part of Lab 6 Exercise 1:

The program sets a start and end time for a lecture at the University. It will ask the user to enter the start time for the lecture, using the 24 hour format (HH:MM:SS). You need to check that the input produces a valid time:

1. Check if the string contains the right characters (i.e. two characters [0-9], a colon, etc.)
2. Check if the time entered makes sense (i.e. HR is 0-23, etc. ... for instance, 33:79:99 does NOT make sense).
3. If the time is incorrect, output an error message and exit the program.

Follow the same procedure for the end time. Once the user has entered two valid times, your program will output the start and end time in a 24 hour format.

Example 1:

```
Enter the start time for the lecture (format is HH:MM:SS): 12:22PM
The start time entered is invalid!
```

Example 2:

```
Enter the start time for the lecture (format is HH:MM:SS): 23:59:59
Enter the end time for the lecture (format is HH:MM:SS): 23:85:00
The end time entered is invalid!
```

Example 3:

```
Enter the start time for the lecture (format is HH:MM:SS): 09:05:00
Enter the end time for the lecture (format is HH:MM:SS): 10:15:00
The lecture starts at 09:05:00 and ends at 10:15:00
```

Example 4:

```
Enter the start time for the lecture (format is HH:MM:SS): 13:00:00
Enter the end time for the lecture (format is HH:MM:SS): 13:15:30
The lecture starts at 13:00:00 and ends at 13:15:30
```

What to hand in

When you are done with this lab assignment, you are ready to submit your work. Make sure you have included the following ***before*** you press Submit:

- Your **Time.h**, **Time.cpp**, **main.cpp**, and a list of Collaborators.