CISP 400 - Lab Assignment #5
Due Thursday, October 4, 2018 at 11:59 PM

For this exercise we are going to make an extremely simple simulation of how a thermostat might control the temperature for a laboratory. Note that this is not going to be anywhere near the real complexity of such a task since that would probably require many thousands of lines of code. It is intended to provide an exercise for you to develop skills at handling programs that use multiple source code files.

PROBLEM DESCRIPTION:

A thermostat used in a corporate lab has a controller for a heating unit. When its desired temperature is set, it checks the current temperature. If the temperature is higher than the desired temperature, it turns off the heating unit. If the temperature is lower than the desired temperature, it turns on the heating unit. After time passes, the thermostat changes the current temperature to the desired temperature. The thermostat must be able to report the current temperature to the user, and at their option, let them set a new desired temperature.

TASK:

Your task is to:

1. Identify the classes of objects.
2. Identify their responsibilities.
3. Draw a UML diagram representing these classes.
4. Write the code for these classes. Use a header file to contain the class definitions and prototypes. Write one class per file, and one "main program" file to run it all.

TIPS:

- You don't need very many classes to make this work. It can be done in 2-3 classes. If you have 4 or more, that's too much.
- There is no ACTUAL time being used in this program. No need for an actual pause, time elapse, current time, time stamps, or anything along those lines. In fact, I would discourage you from using these at all as they may make it more difficult to grade.
- Remember how to write a header file. It is important to make this program work.
- Remember to get the make file sample when writing or testing the program on the Linux server. It is available on the course website. REMEMBER to CHANGE THE PROGRAM NAME in the makefile!
- Remember the constructors for each of these objects, as well as the destructors.
- Remember you can forward declare the classes, if it makes writing them more easy.
- Do NOT make a User class. They are irrelevant to the problem.

DETAILS:

As this is NOT a real thermostat and does not control REAL equipment, we will print out messages to represent the complex tasks required for such systems. For instance, when the heater is turned on, there will be a message printed that tells the user that this has happened.

As we also have no real thermometer, The user should be asked at the start of the program for the current temperature and the current desired temperature, and the program will offer an opportunity to pretend that enough time has passed to make the desired and current temperature the same.

Note that every class should have a constructor that announces an object is being built, and a destructor announcing that it being destroyed shutting down. Also, the way I implemented my own program, I used dynamically allocated objects, so be sure to delete those if need be, either in the destructor (if objects are allocated within member functions and member variables) or at the end of the main program.

Something to keep in mind is that the WAY you do this is EXTREMELY important. It MUST use multiple, interacting classes, which follow the guidelines laid out thus far for good code. There will also be some objects that own others, as per the UML example in class.

Most of the logic is relatively simple. Do NOT turn in a program that is just a main() function accomplishing these tasks or anything similar. Programs that do not even try to comply with this will receive little credit, if any at all.

EXAMPLE OUTPUT:

This assignment is, by its very nature, somewhat difficult to grasp. I have posted example output to give an idea of what I am expecting. Be sure to pay attention to what is going on, and what part of the system is doing what. Each class and subsystem announces what it is as it works.

DELIVERABLES:

Each class should be in a separate .cpp file. The should also be a separate .cpp file that contains main() and any functions that are not part of any class. There also needs to be a header file. So you should have:

1. one file with the .h filename extension that gives a declaration of all classes
2. one file with the .cpp extension that has main() and other functions that do not go with any classes
3. multiple files, each one containing the code for one (and ONLY one) class, so if you have 2 classes you have 2 files, if you have 3 classes you have 3 files, etc.
4. One file containing the UML diagram. This can be in Word format, OpenOffice format, PDF format, or some picture format (.PNG strongly preferred).

The source code files must be able to link together and run as a single executable program to fulfill the requirements specified. INCLUDE THE MAKEFILE.

These files need to be delivered in the .ZIP file format. Do not include the executable files, all I want are source files and the UML diagram.