Fall 2019

ECE 3574 Applied Software Design

Final Examination

10:00am-12:00pm December 16, 2019

THIS IS AN OPEN-BOOK, OPEN-LAPTOP, AND OPEN-INTERNET EXAM.

(NO ONLINE DISCUSSION ALLOWED)

INSTRUCTIONS:

- This is an open-book, open-laptop, and open-Internet examination. You may use your laptop to, for example, refer to your notes, your previous exercises and project source code, web search, and compilation, etc.
- NO discussion with others (regardless of in-person or online). Otherwise, it is considered to be an honor code violation.
- This examination has a total of 4 pages, including the cover page.
- There are three programming problems for a total of 60 points. The exam assumes that you have the course development environment setup correctly on your laptop. You can also use the virtual machine distributed as part of Project 3 to test your code.
- To begin, go to https://canvas.vt.edu/courses/97737/assignments/758609, and download the file final.zip. Unzip the file onto your machine. The zip archive contains three directories, one for each problem below. Complete each problem. Then zip up the source code (no build files!) for all three problems, and submit through the same link on Canvas. This must be completed before the end of the exam (submission will be disabled after the exam ends). Failure to complete uploading the correct zip file by the end of exam will result in no credit being assigned.
- Some questions may be harder than others. Read them all through first and attack them in the order that allows you to make the most progress. Please read each question carefully and code precisely.

Good luck!

Problem 1. (20 points)

Implement a simple mass converter using Qt.

Input	Input unit	Output	Output unit	Relation
kg	Kilogram	lb	Pound	lb = kg × 2.205
kg	Kilogram	oz	Ounce	oz = kg × 35.274
lb	Pound	kg	Kilogram	kg = lb / 2.205
lb	Pound	oz	Ounce	oz = lb × 16
oz	Ounce	kg	Kilogram	kg = oz / 35.274
oz	Ounce	lb	Pound	lb = oz / 16

Table 1. Formulas for mass conversion.

In the above table, whenever the input/output unit is Kilogram, we denote it as kg; whenever the input/output unit is Pound, we denote it as lb; whenever the input/output unit is Ounce, we denote it as oz. For example, if the input is 16 and the input unit is Ounce, i.e., oz = 16 Ounce, then it converts to Pound as follows:

$$lb = oz / 16 = 16 / 16 = 1$$

Figure 1 gives a reference GUI design for the program, where there is a group of **exclusive** radio buttons for specifying the unit for input, and another group of **exclusive** radio buttons for specifying the unit for output. When the pushbutton "Calculate" is pressed, the program will convert the mass in the Input LineEdit and display the result in the Output Label.

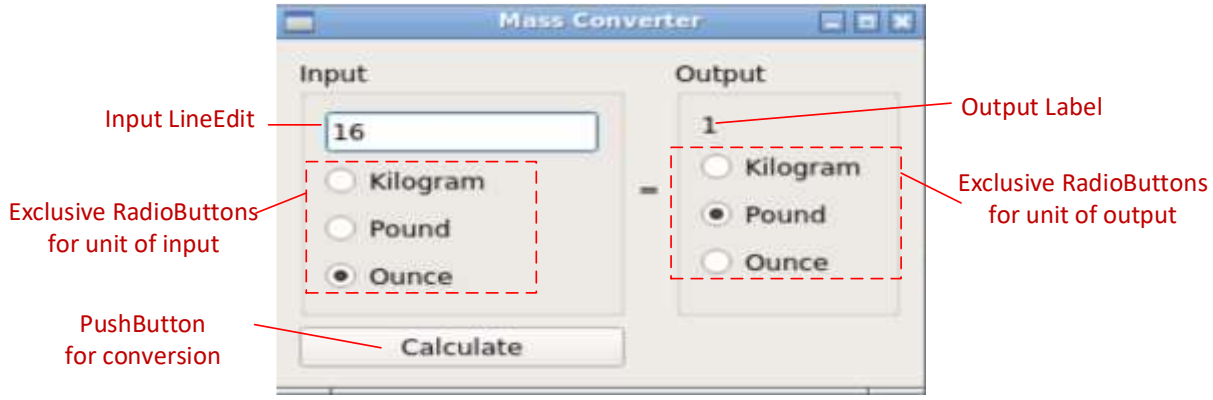

Figure 1. A reference GUI for mass converter.

You should keep 4 digits of precision (4 digits after the decimal point). You may refer to a similar widget from Google (just search "kg to lb" in google.com) to see how it works.

Implement the above program in the three source files `main.cpp`, `massconv.cpp`, and `massconv.hpp`. The included `CMakeLists.txt` file is setup to correctly build the application as an executable named **`P1`**.

Problem 2. (20 points)
Consider a text-based drawing application implemented in `singleton.cpp`. `Shape` is the abstract base class of concrete shapes that can be drawn. `Point` and `Line` are child classes of `Shape`. `Singleton` maintains all created shapes for ease of management. The `main()` function randomly creates four shapes and draw them. It then deletes all created shapes and prints out the number of remaining shapes from the `Singleton`.

1. Implement the constructor
`Singleton (Singleton const&)`
and the copy operator
`void operator=(Singleton const&)`
such that there can only be one instance of the class `Singleton` (10 points).

2. For some reason, the number of printed shapes is not 0 even after deleting all shapes. Fix the bug in the given code (5 points) and explain why your code fixes the bug **as comments** in the source code (5 points). You are not allowed to change `main`().

Problem 3. (20 points)
Consider a class `MyStack` implemented in `my_stack.hpp`. The `MyStack` class has three methods, `push(int)`, `pop()` and `size()`. As is, the `MyStack` is not thread-safe. So if two or more threads concurrently call the methods, the correctness of operations are not guaranteed.

Modify the `MyStack` class in `my_stack.hpp` so that it is thread-safe. If a stack is empty, `pop()` **waits until a valued is added**. `push()` adds a value and then wake up a waiting thread.

To ensure if your modification is correct, write a test case in `test_my_stack.cpp` using the Catch framework. In the test case, four threads should be created. In the test case, two threads insert 1,000 elements each (2,000 elements in total). Two more threads pop 500 elements each (1,000 elements in total). The test procedure then checks if the number of elements in the stack is 1,000 or not. **Implement multi-threading only using in C++11**.

The included `CMakeLists.txt` file is setup to correctly build the test as an executable named **P3**.