COSC 220 - Computer Science II
Lab 9

Dr. Joe Anderson

Due: 12 November 2019

1 Objectives

1. Develop familiarity with function templates
2. Develop familiarity with class templates
3. Practice writing code according to formal specifications and tests

2 Tasks

2.1 In Lab

1. Create a class that will function as a stack, called `MyStack` and make it a template class:

```
template <class T>
class MyStack{
  private:
  struct Node{
    T data;
    Node* next;
  };

  /* Fill in stack operations: push, pop, peek */
}
```

   (a) To help with this, you may adapt some of your code from the `PayRollStack` lab, however you will have to refactor a number of things:
      i. If you were using specific members of the `PayRoll` class, e.g. assigning individual members, you will have to rely on `T` having an assignment operator.
      ii. The destructor will have to rely on the `T` destructor as well. Calling `delete` on a `Node` will implicitly call the destructor of its members, including the `T` inside.
      iii. You may need to adapt any print functions to rely on overloaded `<<` operators instead of calling specific getters.

   (b) You will need to re-structure your `#include` directives slightly to make sure all the necessary template code is generated. As discussed in class, we have seen several different ways to do this:
      i. Include the template definitions in the head files.
      ii. Reverse the `#include` statements so that the `.cpp` files are `#include`'ed from the headers.

iii. Explicitly `#include` the `.cpp` file from the file with `main` and also declare `template class PayRollQueue<PayRoll>` after the `#include`s.

2. Since `MyStack` relies heavily on dynamic memory, be sure to include a copy constructor, assignment operator, and destructor.

3. The `peek` method should just return a *copy* of the `data` element of the node on top of the stack (i.e. just return it by value, relying on the copy constructor).

4. Write a `main` function to create stacks with `PayRoll` objects from previous labs, as well as stacks of `int` and `std::string` objects. Then run some basic tests to be confident the different features of the stack functions.

   (a) You should be able to reproduce the results of Lab 4, by now using a stack of type `MyStack<PayRoll>` intead of a `PayRollStack`.

5. Add comprehensive tests to main by creating `myStack` objects using at least three different data types as the stack values (e.g. `MyStack<int>` and `MyStack<std::string>`). This may include the tests with `PayRoll` and `int`, as above.

6. Include a `Makefile` to compile your code.

3 Submission

Be sure your code conforms to the above specifications. Upload your project files to the course canvas system in a single zipped folder: To zip on Linux:

1. To zip a single folder into a single archive called "myZip.zip":

```
zip -r myZip.zip folderName
```

2. To zip multiple files into a zip file called "myZip.zip":

```
zip myZip.zip file1.cpp file2.h file3 file4.cpp
```

Turn in (stapled) printouts of your source code, properly commented and formatted with your name, course number, and complete description of the code.

Also turn in printouts reflecting several different runs of your program (you can copy/past from the terminal output window). Be sure to test different situations, show how the program handles erroneous input and different edge cases.