Lab5: Stacks and Queues Game[1]

In this lab you are going to begin by looking at the newest Abstract Data Types (ADTs) given to us. You will be working with four different types of containers specifically a stack, queue, priority queue, and mob. Do not worry. You will only be implementing your own version of a stack for this lab as you are implementing most of the other structures in the homework.

The first part of this lab is an exercise in identifying data structures. You won't be graded on this part but it would be useful to spend some time studying these behaviors.

The second part, just to make sure no one is having too much fun, we'll add a few work items that will give you an opportunity to actually build a stack and to get a better feel for what C++ concepts are needed for such a task.

Lab Assignment

[1] As promised, here is a fun guessing game: a program chooses a container for you, but you don't know what it is. A container can be a stack (Last In First Out), a queue (First In First Out), a priority queue (Most "Important" One Out), and a mob (Random One Out). The new container types are the queue, the priority queue and the mob.

For a priority queue, some characteristic of the stored data determines the priority. For example, for a numeric priority queue, the magnitude of the number may determine its priority (hence we can have a Max priority queue or a Min priority queue).

In a mob container the number that comes out is selected at random. A mob is also sometimes called a bag, which means that you can reach into the bag and grab one of the stored items at random.

The object of the game is to guess what container the program selected for you, by just looking at the input and output data. You do not need to submit anything for this task of the lab, so just play and have some fun, but make sure you understand how the four containers work. Get the following files for the game code: RanGen.h, RanGen.cpp, Container.h, and GameDriver.cpp. These can be found in the github repo.

[2] C++ developers have been very nice to you. Inside the STL there is a class called a stack. We are going to use this stack to reverse the characters in a string. For this task create a function called

```
string stringReversal1(string input)
```

[1] *adopted from Dr. Tom Bailey's Lab07

that is responsible for creating a stack of characters (stack<char>) and using the stack to reverse the characters in the string. The methods of the stack you will be interested in are:

(a) bool empty() const;

(b) void pop();

(c) void push(char& c);

(d) char& top();

[3] Now you are going to need to create a new function called

```
string stringReversal2(string input)
```

that is responsible for once again reversing a string of characters. Instead of using a stack to reverse the string, this time you will be using a vector of characters (vector<char>). The methods of vector you will be interested in are:

(a) bool empty() const;

(b) void pop_back();

(c) void push_back(char& c);

(d) char& back();

[4] In this exercise you will be once again creating a new function called

```
string stringReversal3(string input)
```

that is responsible for reversing a string of characters. This time you will be using a list of characters to perform the task (list<char>). The methods you will be interested in this time are:

(a) bool empty() const;

(b) void pop_back();

(c) void push_back(char& c);

(d) char& back();

[5] Your final task is to implement your own stack called MyStack. MyStack will have one data member of type vector<char> and three methods:

(a) bool isEmpty() const;

(b) void push(char& c);

(c) char pull();

Notice you have one method called pull. This method is responsible for both top and pop for our normal stack. Create another function called

```
string stringReversal4(string input)
```

that uses your implementation of a stack and reverses a string of characters. Once you have this done comment out the vector<char> data member and replace it with a list<char> data member with the same name. Notice that you didn't have to change any of the other methods in the class because lists and vectors use the same methods to behave as a stack. The class you have created (MyStack) is an example of the facade design pattern. It provides a scaled down set of methods to the user to perform more specific tasks.

Turn in:

You should have 3 files to upload. Make sure your files are named Lab7Driver.cpp, Lab7MyStack.h, and Lab7MyStack.cpp and that your name is IN COMMENTS AT THE TOP OF EACH FILE. As always labs are due 1 week past their assigned date.