**CS311 Yoshii - Homework 1 Part 2 Vector Stack (based on Week2)**

**===============================================================**

**Due: Week 4 Tuesday at the beginning of class**

**Total: 23 points Your score is:**

**Your name:** Justin Goulet

**Date submitted:** Sept 21 2015

**===========================================================**

**Purpose: To appreciate the fact that we can use a different implementation of a stack.**

===========================================================

**Make sure it works with g++**

**Problem Analysis [5pts]<separate sheet> Your score:**

Do the problem (Problem Analysis 1 or 2) you did not do in class. This time, do it by yourself.

//This was the Elevator, I am assuming?

The elevator is a marvelous invention. It allowed the immediate flop of wealth from the floor to the roof, causing the most expensive real estate in the world to be suspended with sky scrapers.

There is plenty of logic around the elevator: when to stop when to accelerate, decelerate, and when to open the doors. Let’s look at how ew can get ours to work.

For my example, I wan to look at the Empire State Building. Efficiency is key here as the tower is quite tall. There are several allocations of elevators throughout the building, all located in the center. Each is based on which floor you are heading to. For maximum efficiency, we are going to split the building vertically in 2. One set of elevators travel tot he first half of the floors, the other, the other half. This is to split up the allocation and dramatically decrease the wait time for patrons.

Each of the elevators will now act in a similar way. The functionality is key, as we don’t want people to hate going up and down in this monster. The elevator will always resort to the lowest floor, however, will also track which floors it previously visited, and the usual times in which people leave. The inside buttons have priority on which floor the elevator stop sat, with direction also factored in.

If the elevator is going up, it will be at quite a large speed. We will have to look 5-10 floors up at the buttons to see if they were tapped to see if the elevator needs to stop. They are ignored if they are going the opposite direction as well.

Back to remembering the history. We want the elevator to be efficient, so we are going to try to remember and learn patterns from the day by day basis. We are going to collect information about what times the elevator usually arrives at a floor, and eventually learn when the elevator needs to arrive in future times. This means that as soon as it learns the patterns of those working inside the building, the wait time will virtually vibe zero for the regular workforce-our main priority.

As for safety, there will need to be plenty of brakes in the shaft of the elevator as well. An emergency brake inside will be helpful, as well as an emergency phone that operates by battery. If, for some reason, a cable breaks while transporting passengers, we will need to install a safety lock inside the shaft that will work as a gear break for the elevator, when it detects an unusual reaction. We can attribute this to the original innovator back at the wolds fair in the 1930’s.

**Questions: [3pts]<answer here> Your score:**

Q1. Name 3 types of information found in an activation record of a function.

Parameters, if any

Location in memory in which the original class ended

Location in memory where the method is

Q2. What is the name of the stack that holds activation records?

Runtime Stack

Q3. Where are dynamically allocated data structures found?

They are found in the runtime stack because that is when they are used.

**Programming - Vector-based Stack: [15pts] Your score:**

Re-write the HW1P1 stack class (**copy and then rename as vstack.h and vstack.cpp**) so that

* It now uses a **vector** instead of an array.
* There is no maximum size now. The stack **starts out having no slots** and it will grow as elements are added.
* N o need for top. Take it out. You can call size()-1 to get top.
* Destructor has to do some work to make sure it leaves no cells behind.
* Constructor has no work to do.
* isFull always returns false.
* Pop\_back() and front() do not return the top item which is at size()-1.
* Client: in case of error, destructor will not be called but OK.

Now, use this stack class with the **HW1P1 client program** to evaluate post-fix expressions. **Test with the same cases as before.**

**HW1P1 client should not be changed except to include the new header file. Your new implementation should be invisible to the client.**

**Q1) Did you notice any different results from HW1P1 test results?[1pt]**

**No, The transition was pretty smooth in the client program. Just had to change the name of the data structure.**

**Q2) The state of the program statement : [2pts] <answer here>**

* **Does your program compile without errors?**
  1. **Yes, My program works without any errors**
* **List any bugs you are aware of, or state “No bugs”:**
  1. **Except for the queue (commented out), there are no bugs in my current program. (That I am aware of)**

**SUBMIT THESE FILES TO COUGAR COURSES:**

**Always make sure the files you submit can be opened on a PC**

**Always include the word Client in the client file name.**

**Always start the name of the script/screen dump with Test.**

1. **This assignment sheet with your inserted answers.**
2. **Problem Analysis sheet with your answers.**
3. **The source file(s) (header, implementation, and client) with good comments. Always check against the How to Comment file.**
4. **The script or screen dump of compiling and executing your program.**

**Whether working or not, test result must include the lines for compiling your files or we will not grade our program i.e. 0 points for the program.**

**Did you check your comments and style against CS311 How To Comment.doc??**