CSC 2431 Assignment 3
Due: Friday 5/6/16** by 11:59pm

****Note 5/6 is the day of the mid-term. You are being given more than 1 week to work on the lab, I would highly recommend you start early.**

Part 1 – Textbook Exercises

There are no required textbook exercises for this assignment. In preparation for the exam, I would recommend picking some exercises from each relevant chapter for extra practice.

Part 2 – Programming Assignment

In computer architecture, most machines are *register-based*, and have assembly instructions that take the following form:

```
add X, Y, Z
```

which means "add Y and Z and put the result in X." The operands to the ADD instruction (X, Y, and Z) are stored in registers – small, ultra-fast storage for a single piece of data. In contrast, a stack-based machine uses a stack to keep track of operands. For example, the equivalent code for ADD X, Y, Z in a stack machine could be:

```
push Y
push Z
add
pop X
```

In this case, X, Y, and Z are locations in memory. This code pushes the data at memory location Y to the top of the stack, and then pushes Z on top of that. The `add` instruction then adds Z and Y and puts the result in X. Your goal for this assignment will be to create an emulator for a simple stack machine with instructions of this form. Your emulator will provide the following instructions:

`push N`
-this instruction pushes a value from memory at location N onto the top of the stack

`pop N`
-this instruction pops from the top of the stack into memory location N

`add, subtract, multiply, divide`
-Each instruction pops the top two elements off the stack (assume the top element is the right-most operand to the operation), performs the desired operation, and puts the result on the top of the stack.

The stackMachine Class

You will create a class with the following properties:

1) A private member: a fixed-size, statically allocated array of 100 elements for the stack machine's "memory." *The type of elements to be stored will be defined via a template, assumed to be a numeric type.*
2) A private member: an STL stack to keep track of the operands for the stack instructions.
3) A single, default constructor to initialize both private members appropriately (the stack should remain empty, the array should contain all zeroes).

4) A method for each of the instructions described above (push, pop, add, subtract, multiply, divide).
    a. Make sure to do bounds-checking on the array (e.g., can't pop to an array index less than 0 or greater than 100) and stack (e.g., can't call add on an empty stack).
5) A method to read instructions from a file, line-by-line, and call the appropriate methods (see below for example, test files are on Blackboard).
6) A method to retrieve an element from memory, indexed by position.

Testing

To test your emulator, you will load two files with sample assembly language programs and execute each instruction (i.e., call the corresponding class methods), line-by-line. After executing the last instruction, print the entire contents of the stack machine's memory, 10 elements per line (each element separated by a space). The assembly language programs have the following format:

```
.data
<data in memory, separated by spaces, undefined are assumed to be 0>

.text
<code goes here>
```

For example, the following code adds 0 through 5 and puts the result back in memory at location 0:

```
.data
0 1 2 3 4 5

.text
push 0
push 1
push 2
push 3
push 4
push 5
add
add
add
add
add
pop 0
```

What to turn in:

1. Your header (.h) for the class, and source for your test client (.cpp).
2. Your output after running the two test files in your client.
3. All documents should be uploaded to Blackboard following the assignment submission instructions (linked on Blackboard).