CPSC 2120/2121
Lab 04

Stack-based calculator

Due date/time are on the CPSC 2121 Canvas website

Learning Objectives

- To improve our ability to implement algorithms and data structures in C++
- To use a simple data structure (a stack) as a building block to construct more complex system
- To become proficient in fundamental data structures used throughout computer science
- To improve our analytical skills while gaining familiarity with mathematical tools used in the analysis of algorithms
- To implement a data structure to understand its intricacies

Problem / Exercise

For this lab, you will implement a six-function calculator using stacks. This calculator will read input in using standard infix notation and output the correct calculated value. Your calculator must make use of parentheses to determine order of operation. Division of your program into appropriate functions is strongly recommended. In addition to the four standard operations of addition, subtraction, multiplication, and division; you will implement exponentiation and modulus. You are free to use the stack class from the C++ STL and any of its functions for this assignment. Your calculator must be implemented in a file called Calc.h (not provided).

Additional Requirements

- Your Calc class must compile and run with unmodified versions of the provided lab04.cpp and Makefile files to produce the same output as our examples. After getting your class working with our test program, you should create several further tests on your own (which you will submit with your code)
-

Examples

Your C++ source code should be in a file called Calc.h, and it should be compiled (along with our provided lab04.cpp file) into an executable called lab04.out using our provided Makefile. The output of our testing program with your class implementation must look exactly like the examples below when run on the command line of our Unix machines.

```
./lab04.out

((2+2)-2)*2
4
(1+2)^2*((3-4)/5)
-1.8
(4^3)%3
1
(1+2)
3
5-2
3
81/27
3
5^0
1
16%7
2
```

Source Code Requirements

- Put a comment at the top of your source code file(s) with your name (first and last), the date of your submission, your lab section, and the assignment's name.
- All functions should be commented. Use inline documentation, as needed, to explain ambiguous, tricky parts, or important portions of your code.
- All source code must follow good programming style standards such as properly indenting source code and all variables, functions, and classes should be well-named.
- Your class must use dynamic memory allocation and deallocation properly, which means your class cannot contain a memory leak. You may use valgrind to check for memory leaks. Refer to the Unix manual and valgrind's online documentation for more information on valgrind.

Submission

Before the deadline given by your lab instructor, you need to submit the code to your CPSC 2121 Canvas webpage under the correct lab assignment. Make sure to submit all of the following.

1. All source files required for this lab (Calc.h)
2. Unit tests (lab04.cpp)

After you submit, always double check that the file(s) you submitted were the correct version. To double check, download the submitted file(s), put them on one of our Unix machines, and make sure they compile and run correctly.

Grading: 10 points

If your class does not compile on our Unix machines (or your assignment was not submitted on time), then you'll receive a grade of 0 on this assignment. Otherwise, your class will be graded using the criteria below.

Your class works correctly with a program we will use to test your class	6 points
Proper variable names, documentation, and code organization	2 points
Your class uses dynamic memory allocation/deallocation properly (no memory leaks)	1 point
At least 3 distinct, passing test cases (not included ones provided) are submitted	1 point
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly and efficiently on our Unix machines with any given set of valid inputs. This means that you need to create many examples on your own (that are different than the aforementioned examples) to ensure you have a correctly working class. We will only test your program with valid sets of inputs.