CPSC 2120/2121
Lab 08

Infix to Postfix and Postfix Evaulation using stack based algorithms
Due date/time are on the CPSC 2121 Canvas website

Learning Objectives

- To improve our ability to implement algorithms in C++
- To implement an algorithm that uses a fundamental data structure (a stack)
- 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

Problem / Exercise

For this lab, you will implement an algorithm that will convert an input infix expression to a postfix expression as shown in the examples section. Also, you will implement an algorithm that will evaluate the postfix expression found by the previous algorithm as shown in the examples section. Both algorithms must use a stack from the C++11 STL, and you may use any additional data structures from the C++11 STL. You will implement your algorithms in a file called lab08.cpp (not provided), and this program must accept a single command line argument, argv[1], which will be an input infix expression that has the following properties.

- The input will be a single string consisting of tokens, where each token will be separated by a single space so we can parse the input in an easy manner.
- Each token in the input is either an operand (of type double) or an operator. The operators for this lab are left parenthesis, right parenthesis, +, -, *, and /, and the normal C++ operator precedence rules for these operators must be handled by your algorithms.
- +, -, *, and / will only be used as binary operators (they will not be used as unary operators).
- You may assume the input will be a syntactically correct C++ arithmetic expression that follows the specifications aforementioned.

Parsing Input

There are many ways to parse string inputs in C++, and one such way is shown below. The code below will parse an input string for this lab assignment (based on the specifications aforementioned) and print out each token to standard output. You may use and modify this code for your lab assignment.

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

//This program will print tokens in argv[1] to stdout
//if each token is separated by a single space
```

```
int main(int argc, char * argv[]){
  string token;
  string infix = argv[1];
  stringstream ss(infix);
  while( getline(ss, token, ' ') ){
    cout << token << endl;
  }
  return 0;
}
```

Additional Requirements

- Your program must compile and run with an unmodified version of our provided Makefile to produce the same output as our examples. If your program does not, then you'll need to fix your program. After you get your program working with our examples, then you should create several examples on your own to further test your program to make sure it works with any valid set of test cases.
- Your program must use a C++11 STL stack in its implementation. Also, you may use any other data structures in the STL (hint: an output list of strings might be useful to have along with your stack).
- Your postfix expression output must contain a single space between each token as shown in the examples.
- When you finish this assignment, on a piece of paper, analyze the run time and space complexities of your algorithms.

Examples

Your C++ source code should be in a file called `lab08.cpp`, and it should be compiled into an executable called `lab08.out` using our provided `Makefile`. The output of your program must look exactly like the examples below when run on the command line on our Unix machines. Each example is a separate run of a complete program, and the command line argument, argv[1], is a string enclosed in single quotes. The single quotes are not considered as part of the input (they are not tokens), and single quotes are used in order to pass a single string containing spaces from the command line to your program.

```
./lab08.out '6 + 4 * 2.5'
Postfix Expression: 6 4 2.5 * +
Postfix Evaluation: 16

./lab08.out '( 6 + 4 ) * 2.5'
Postfix Expression: 6 4 + 2.5 *
Postfix Evaluation: 25

./lab08.out '( ( 6 + 4 ) * 2.5 )'
Postfix Expression: 6 4 + 2.5 *
Postfix Evaluation: 25

./lab08.out '( 19.5 - 8 ) * 2 / 10 + 6'
Postfix Expression: 19.5 8 - 2 * 10 / 6 +
Postfix Evaluation: 8.3

./lab08.out '5.75 * ( ( 23.9 - 8.9 ) + 2 ) * 9 / 7.5'
Postfix Expression: 5.75 23.9 8.9 - 2 + * 9 * 7.5 /
Postfix Evaluation: 117.3

./lab08.out '( 6 + 7.5 ) * ( ( 2 - 1.5 ) + 2.5 * 4 )'
Postfix Expression: 6 7.5 + 2 1.5 - 2.5 4 * + *
Postfix Evaluation: 141.75

./lab08.out '( 5.5 + 3 ) * 2 - ( ( 0.5 / 0.25 ) + 39 * 10.5 - ( 3 + 2 + 4 ) )'
```

```
Postfix Expression: 5.5 3 + 2 * 0.5 0.25 / 39 10.5 * + 3 2 + 4 + - -
Postfix Evaluation: -385.5
```

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 program must use dynamic memory allocation and deallocation properly, which means your program 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 date/time stated on the CPSC 2121 Canvas webpage, you need to submit your code to our 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 (lab08.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 program does not compile on our Unix machines, your assignment was not submitted on time, or your program did not use a stack, then you'll receive a grade of 0 on this assignment. Otherwise, your program will be graded using the criteria below.

For various test cases, program correctly converts input infix expression to postfix	5 points
For various test cases, program correctly evaluates the postfix expression	5 points
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 program. We will only test your program with valid sets of inputs.