Programming Assignment 3
CSCI 235
Ilya Korsunsky
Due: 4/24/2015 (midnight)

100 points

Follow the instructions presented in the Programming Rules documents. Submit the header and source files as well as your Makefile. Also submit the input file you used for testing. Note that your program needs to compile in order to get any credit at all.

Adhere to the style guidelines.

Start Early!

Algebraic Calculator

You are going to implement a simple calculator. Your calculator is going to parse infix algebraic expressions, create the corresponding postfix expressions and then evaluate the postfix expressions. The operators you are allowed to use are: +, -, * and /. Your operands are integers. The input to your program is a file *infixFile* that contains a sequence of infix expressions (one expression per line). Your output is going to consist of one file: a *resultFile* that contains the postfix expressions and their evaluation.

Part 1 (%60)

You need to read infix expressions from the input *infixFile* (one expression per line) and produce the corresponding postfix expressions that you will save in file *resultFile*. An example of an infix expression is the following:

(12 + 3) * (9 – 74) + 34 / ((85 – 93) +(3 + 5) * 3) - 5

You have to parse this expression from left to right, identify the operands, the operators and the parentheses and ignore blanks. You will implement the infix to postfix conversion algorithm using a stack of operators and parentheses. Use the pointer-based stack implementation we used in class.

You should be able to handle syntactically wrong infix expressions using exceptions. In this case you should output a message on the *resultFile* indicating the type of error (i.e. “missing right parenthesis”).

Part 2 (%40)

Now you need to evaluate each postfix expression using the stack algorithm described in class and place the results in the file *resultFile*.

Extra Credit (10 points)
Our algorithm transforms an infix to a postfix expression according to a left to right association rule when the operators have the same precedence
(i.e. 5+4+3 becomes 5 4+3+ and not 5 4 3 + +).
The exponentiation operator though should follow a right to left association rule. When we write 2^3^5 we mean 2^(3^5) which corresponds to the postfix expression
2 3 5 ^ ^. Our algorithm though provides as a result the expression 2 3^ 5^ which corresponds to the infix expression (2^3)^5. Modify the infix to postfix algorithm so that the exponentiation operator is right to left associative.

Coding Instructions

Create a class (not templated) named InfixToPostfixCalculator. For this class use a header file named InfixToPostFixCalculator.h and an implementation file named InfixToPostfixCalculator.cpp.
This class needs to have one no-parameter constructor and two public functions. The first one is
string ConvertInfixToPostfix(const string &input_infix)

It reads a string holding the infix expression, calculates the postfix expression and then returns the postfix expression as a string.
In the output expression separate the various parts using a comma (,). So for example the string corresponding to a valid postfix expression could look like:

3,134,76,+,-

Note that there are no parentheses in the output postfix expression.

The other function is

double CalculatePostfix(const string &input_postfix)

It reads an input postfix expression (in the format described above), calculates the result and returns it.

Both functions should throw exceptions when the input expressions are not of the right form.

Add as many private functions and data members as necessary.

In the main file you will create an object of type InfixToPostfixCalculator. Then you will read the input file line by line. Each line will be a string that you will pass to ConvertInfixToPostfix(). You will save the result to the result file. Finally, if ConvertInfixToPostfix() returns without an exception, you will run the CalculatePostfix() function on the returned string. You will write the result on the output file as well.

In conclusion you need to have the following files:

(a) The files required for the stack (StackInterface.h, LinkedStack.h, LinkedStack.cpp) and its exceptions: PreconditionViolatedException.h, PreconditionViolatedException.cpp).

(b) The InfixToPostfixCalculator.h and InfixToPostfixCalculator.cpp files as well your exception files (not required if you use the same exceptions as the stack).

(c) A main file Test.cpp that includes the InfixToPostfixCalculator.h and tests the algorithms as described above.