Fall 2018 CSCI 232 Data Structures and Algorithms
Phillip J. Curtiss, Assistant Professor

Project: Infix Algebraic Expression Evaluation Due Date: 2018-10-29 Midnight

Description:

Based on Chapter 6, Problems 6 & 7; write a program that accepts an infix algebraic expression, converts it to a postfix expression, and then evaluates the postfix expression. Prompt the user for the values of the operands and evaluate the expression. The program should prompt and produce output similar to:

```
Enter the infix expression: a + b * (c - d) + e / f
Enter the values for the operands in a + b * (c - d) + e / f:
a: 1.5
b: 2
c: 3
d: 4
e: 5
f: 6
The postfix expression is: abcd-*+ef/+
The value of the expression is: 0.333333
```

You have been provided the Node and LinkedStack classes, as well as the PrecondViolatedExcep custom exception class. You must implement the Postfix class which must at least implement the methods `getInfix()`, `getPostfix()`, and `getResult()` as shown in the provided header and implementation files.

Specific Project Tasks:

The following tasks need to be performed in order to complete the project:

1. Implement the Postfix Class by filling out the provided header and implementation files.
2. Store your infix and postfix expression using strings (use the C++ string class).
3. Test your program on actual algebraic expressions.
4. You should handle invalid expressions if characters that are not letters or `+, -, *, / and ^` are entered. (Note that `a^b` is `ab`).
5. If the expression is invalid (unbalanced parentheses, missing operators, etc), then print an error message.
6. if the expression uses an operand multiple times only prompt the user once for that operand. For example, if the expression `c+b*c` is entered, only prompt the user one time for c.
7. Allow embedded spaces in the infix string.
8. For an extra 10 points, make use of Smart Pointers in your solution - replacing raw pointers used in provided code.

Obtaining Project Files:

1. Log into your account on `gitlab.cs.mtech.edu` and fork the project for today's lab `Project-1`.
2. Add your instructor as a `Developer` member to the `Project-1` project you just forked into your account.

3. Change working directory on `lumen` and obtain the files for the lab by executing the following commands:

   ```
   mkdir Projects
   cd Projects/<lab project>
   git clone <project spec ssh url>
   git status
   ```

Figure 1: Programming Project Grading Rubric

Attribute (pts)	Exceptional (1)	Acceptable (0.8)	Amateur (0.7)	Unsatisfactory (0.6)
Specification (10)	The program works and meets all of the specifications.	The program works and produces correct results and displays them correctly. It also meets most of the other specifications.	The program produces correct results, but does not display them correctly.	The program produces incorrect results.
Readability (10)	The code is exceptionally well organized and very easy to follow.	The code is fairly easy to read.	The code is readable only by someone who knows what it is supposed to be doing.	The code is poorly organized and very difficult to read.
Reusability (10)	The code could be reused as a whole or each routine could be reused.	Most of the code could be reused in other programs.	Some parts of the code could be reused in other programs.	The code is not organized for reusability.
Documentation (10)	The documentation is well written and clearly explains what the code is accomplishing and how.	The documentation consists of embedded comments and some simple header documentation that is somewhat useful in understanding the code.	The documentation is simply comments embedded in the code with some simple header comments separating routines.	The documentation is simply comments embedded in the code and does not help the reader understand the code.
Efficiency (5)	The code is extremely efficient without sacrificing readability and understanding.	The code is fairly efficient without sacrificing readability and understanding.	The code is brute force and unnecessarily long.	The code is huge and appears to be patched together.
Delivery (total)	The program was delivered on-time.	The program was delivered within a week of the due date.	The program was delivered within 2-weeks of the due date.	The code was more than 2-weeks overdue.

The *delivery* attribute weights will be applied to the total score from the other attributes. That is, if a project scored 36 points total for the sum of *specification*, *readability*, *reusability*, *documentation* and *efficiency* attributes, but was turned in within 2-weeks of the due date, the project score would be $36 \cdot 0.7 = 25.2$.

Project Grading:

The project must compile without errors (ideally without warnings) and should not fault upon execution. All errors should be caught if thrown and handled in a rational manner. Grading will follow the *project grading rubric* shown in figure 1.

Collaboration Opportunities:

This project provides *no collaboration opportunities* for the students. Students are expected to design and implement an original solution specific in this project description. Any work used that is not original should be cited and properly references in both the code and in any accompanying write-up. Failure to cite code that is not original may lead to claims of academic dishonesty against the student - if in doubt of when to cite, see your instructor for clarification.