EECS 268: Spring 2016

Laboratory 6

Due: This lab is due before your next lab begins.

In this lab, you will develop a stack-based postfix expression validator. Your implementation will be required to catch and handle exceptions as a way of detecting some types of syntax errors.

Details

Your program will prompt for and read a series of strings from the console and test whether each represents a valid postfix expression. The strings will have no embedded blanks and will contain some combination of the standard operators (+, -, *, /) and identifiers (single alphabetic characters that are to be treated, like C++ and Java, as being case-sensitive). If you detect any other character during parsing, reject the string as syntactically invalid. Continue prompting for strings until the user enters a single # at the prompt. A sample session with your program might look like (user input shown in ***bold italic***):

```
Enter a string: AB/cw+*
You entered a valid postfix string. The equivalent infix: ((A/B)*(c+w))
Enter a string: XYZ-
You entered an invalid postfix string: it is missing one or more operators
Enter a string: XY-*
You entered an invalid postfix string: it is missing operands
Enter a string XY*BC+C3*++
You entered an invalid postfix string: it contains the illegal character '3'
Enter a string XY*BC+CD*++
You entered a valid postfix string. The equivalent infix: ((X*Y)+((B+C)+(C*D)))
Enter a string: #
```

Notice that the output for valid postfix strings will include the equivalent infix string with parentheses. See below for details on how this will be accomplished.

If a string has multiple errors, you need only detect and report the first.

You can use the templated `Stack` class you created for lab 5, but make sure the maximum size of the array is at least 50. **You are not allowed to add any public methods to the `Stack` class that are not in the `StackInterface` definition in the textbook**.

The instantiating type parameter for your templated Stack will be `std::string`. With each character of the input you parse, you will either push it onto the stack (if it is a valid operand), or (if it is a valid operator) peek and pop two elements, putting the operator between them, and pushing the result surrounded by parentheses back onto the stack. *Be sure you get the operands in the right order (e.g.: A/B in the first user input example above rather than B/A)!*

Do not try to prevent stack exceptions from being thrown (e.g., by using the `isEmpty` method). Instead, "`catch`" them, issue the appropriate message in the `catch` block, and then continue on. (That is, prompt for the next string.) Note that not all postfix expression errors will result in exceptions being thrown, so don't forget to look for and handle those that are not caught that way.

Grading Criteria

Grades will be assigned according to the following criteria:

- Correct reporting of valid postfix expressions and their equivalent infix: 30%
- Correct use of `try-catch` for detecting some types of errors: 25%
- Correct use of other parts of the `StackInterface` for detecting the other types of errors: 25%
- Managing the interactive prompts and creation/deletion of the `Stack` objects: 10%
- Style, including programming style, paragraphing conventions, and internal documentation: 10%

Submission

Once you have created the tarball with your submission files, email it to your TA. The email subject line must look like "[EECS 268] SubmissionName" as follows:

```
[EECS 268] Lab 06
```

Note that the subject should be exactly like the line above. Do not leave out any of the spaces, or the bracket characters ("[" and "]"). In the body of your email, include your name and student ID.