Notation Converter

COP 4530 Programming Project 2

Instructions

For Programming Project 2, you will implement a Deque (Double-Ended Queue) and use that data structure to write a class that can convert between the three common mathematical notation for arithmetic.

The three notations are:
Postfix (Reverse Polish) Notation:
      Operators are written after operands A B - C + == (A - B) + C
Infix Notation:
      The standard notation we use where the operator is between the operands.
Prefix (Polish) Notation:
      Operators are written before operands: * - A B C == (A - B) * C

The converter will be able to convert from one of those three to any other. The input will be a string written in a single notation, and the output will be the conversion to the specified notation.

The input string for the functions will only ever contain the standard four arithmetic operators (* / + -), an operand denoted by a letter (upper or lower case), a left or right parentheses (only round), or spaces. Whitespace (one or more characters of space) separate all operand and operators, while parentheses must have whitespace on the outside (between operators), and inside parentheses whitespace is optional. Parentheses are only used in infix strings. Parentheses must have operators between them, as all operations must be binary and not implied. For example:

```
Valid Postfix: c  d / a b * r r       * / *
Valid Prefix: *   +    A B - C D
Valid Infix: (( X + B ) * ( Y - D ))

Invalid Postfix:c d a b * r r * / * (Operators don't match up with operands)
Invalid Prefix: * ^ A B & C D (Invalid Characters)
Invalid Infix: ((a / f) ((a * b) / (r * r))) (No operator between
parentheses)
```

The output string should always separate all operand and operators by ONLY one space. The interior of a parenthesis should have no whitespace between the letter and the parenthesis or another parenthesis, while the exterior of a parenthesis should be separated by ONLY one space from an operator and none for another parenthesis. For example:

```
Valid Output: ((x / y) - g)
Valid Output: ((x / y) - (a * b))
```

Valid Output: `x y * g / h +`

If your output does not conform to this standard, you will not pass the tests required for this project.

Again, there is an abstract class for you to inherit that has you implementing the following methods.

Abstract Class Methods

std::string postfixToInfix(std::string inStr)
This method takes in a string of **postfix** notation and returns a string in the **infix** notation

std::string postfixToPrefix(std::string inStr)
This method takes in a string of **postfix** notation and returns a string in the **prefix** notation

std::string infixToPostfix(std::string inStr)
This method takes in a string of **infix** notation and returns a string in the **postfix** notation

std::string infixToPrefix(std::string inStr)
This method takes in a string of **infix** notation and returns a string in the **prefix** notation

std::string prefixToInfix(std::string inStr)
This method takes in a string of **prefix** notation and returns a string in the **postfix** notation

std::string prefixToPostfix(std::string inStr)
This method takes in a string of **prefix** notation and returns a string in the **postfix** notation

These methods will all be instance methods of the class NotationConverter (which must be called "NotationConverter"). You will also need to implement a fully function deque using a doubly linked list. When writing the methods to convert between the notations, you can **ONLY** use your deque and strings for storing data in the algorithms. You may not use any C++ Standard Library containers, such as the STL Deque, Stack, Queue, Vector, or List.

This project will be tested using the Catch2 (https://github.com/catchorg/Catch2) test framework. This framework only requires that a program include the header to run the test file. The test file that will be used to grade the project is included and can be used to test your code before submitting. For your convenience, this project contains the Catch2 header. The course documents include a quick tutorial on how to use Catch2 and test files. I require and expect that you keep the Catch2 header in your project directory at that directories base.

Examples

Below are some examples of how your code should run. The test file can also be used to get an idea of how the code should run.

```
NotationConverter nc;

std::string examplePost = "c d / a b * r r * / *";
nc.postfixToInfix(examplePost)  // Infix: ((c / d) * ((a * b) / (r * r)))
nc.postfixToPrefix(examplePost) // Prefix * / c d / * a b * r r

std::string examplePre = "* + A B - C D";
nc.prefixToInfix(examplePre)  // Infix: ((A + B) * (C - D))
nc.prefixToPostfix(examplePre) // Postfix A B + C D - *

std::string exampleInfix = "((a / f) ((a * b) / (r * r)))";
nc.infixToPostfix(exampleInfix) // Postfix: a f / a b * r r * /
nc.infixToPrefix(exampleInfix)  // Prefix / a f / * a b * r r
```

Deliverables

Please submit complete projects as zipped folders. The zipped folder should contain:
NotationConverter.hpp (Your header file for NotationConverter class)
NotationConverter.cpp (Your source file for NotationConverter class)
PP2Test.cpp
catch.hpp
NotationConverterInterface.hpp

And any additional source and header files needed for your project. Best practice is to put the code for your Deque and Doubly Linked List in their own header/source files and include them using the preprocessor include in the appropriate files.

Hints

It would be a good idea to decode the input string and place it into a deque to make it easier to read for the various methods

In the algorithm header, there is a method that can reverse strings.

When dealing with infix notations, remember that operator precedence is very important

Remember, a deque can operate like a stack or a queue, queues and stacks can be considered specializations of a deque

You may not need to write a separate algorithm for all six methods. For example, when going from postfix to prefix, you can go from postfix to infix and infix to prefix if those methods are already implemented.

Rubric

Any code that does not compile will receive a zero for this project.

Criteria	Points
Infix notation should convert to prefix notation	10
Infix notation should convert to postfix notation	10
Prefix notation should convert to postfix notation	10
Prefix notation should convert to infix notation	10
Postfix notation should convert to infix notation	10
Postfix notation should convert to prefix notation	10
Checks for invalid characters	5
NotationConverter uses a student implemented Deque	17.5
Code uses object oriented design principles (Separate headers and sources, where applicable)	7.5
Code is well documented	10
Total Points	**100**