COP 3503 – Project, Stage 1

Purpose

This stage involves recursion and recursion-like reasoning in coding while also actively preparing a critical component necessary for the class project.

Description

As mentioned in class, computers don't natively understand mathematical expressions in the way we prefer to write them. While we prefer "infix" notation, where the operators come between the values being operated upon, computers prefer either "prefix" or "postfix" notation, as these are unambiguous and easier for them to operate upon. As the project, at its core, is to write a mathematical calculator, we will need to convert math expressions to one or the other at some point.

The requirement of this project stage is to implement the Shunting-Yard algorithm (http://en.wikipedia.org/wiki/Shunting-yard_algorithm) as it applies to the overall design of this project. You do not need to calculate any values – the goal is to simply turn any input infix expression into a postfix expression, an analysis that will aid the later stages of this project.

You may assume that there will always be spaces between any values and operators, as this can aid parsing and reduce the complexity of your input code.

Example input/output:

```
> 67 + ( 32 / 8 ) * 3 rt 8

67 32 8 / 3 8 rt * +
```

This is all we're looking for in this project stage – no bells or whistles needed. Please keep the prompt as you see it in this example – just a leading ">" to request input – as this will aid us in our grading efforts.

Some quick web searches can easily find you example code for this problem online. Keep in mind that their examples will not likely fully apply to our project spec (use of "rt" as an operator) and will likely need further modification, so whatever you choose to use, you're responsible for knowing how it works and making any modifications necessary for this stage's implementation. You *will* be reusing this code again in the project's final stage.

Secondly, while such online examples do exist, you may find it a worthwhile exercise to attempt your own implementation from scratch, as the necessary logic provides great programming practice. I personally believe you would be best served by referencing the Wikipedia article's explanation and translating that into code.

Submission

Your code and any accompanying documents should be submitted online through the Canvas website's assignment tab for this homework. All submission data **must** be submitted compressed within a single *.zip archive – the assignment tab will be configured to accept only *.zip files.

Your code must be in the base directory inside of your submitted *.zip in order for us to efficiently grade it. Additionally, you must submit a makefile and your *h and *.cpp files – your original source. Should you choose to include any *additional documents*, please put them in one of the following formats *within* the *.zip:

- *.pdf
- *.txt
- *.doc / *.docx

Note that failure to follow these instructions will likely result in an extremely low grade.

Again, please submit a makefile for this project; we'll provide a baseline, single-file makefile for you to use which will require edits if you use a different filename or spread your code across multiple *.cpp files.