Reverse Polish Notation Calculator

Re-submit Assignment

Due Mar 2 by 11:59pm **Points** 100 **Submitting** a file upload
Available Feb 16 at 9am - May 2 at 11:59pm 3 months

Implement the RPN calculator pseudo-code supplied in the lecture as real code. Your program must:

1. Prompt the user for an infix math problem.
2. Convert the problem to postfix.
3. Output the problem in postfix.
4. Calculate the result.
5. Display the result.
6. Ask the user for another math problem.

If the user enters *quit* for the problem, end the program.

Use the standard stack and queue class/methodology provided by your preferred language's framework, such as the STL stack/queue classes in C++ or the Array class in JavaScript. Put the conversion procedure in its own function. Put the calculate result procedure in its own function as well.

You need to handle multi-digit numbers even though the sample code does not. In your version, numbers (operands) are separated from operators by zero or more spaces, while numbers are separated from other numbers by *one* or more spaces. While negative numbers cannot be input by the user, the result may be negative based on the input math problem.

You must support +, -, *, /, and % operators, as well as (potentially nested) parenthesis. For +10 extra credit, also support raising a number to a power with the POW operator, which must appear as those three letters in all uppercase.

Note: You must not submit your "node_modules" folder if you are working on NodeJs/JavaScript. (Just submit your JavaScript source code and package.json file)

RPN Calculator

Criteria	Ratings		Pts
Prompt user for infix expression. Intuitively, expression will be entered in a single line: -3 points if program does not accept single-line expressions.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
Infix -> postfix conversion. Resulting postfix expression must be logged to console.	15.0 pts Full Marks	0.0 pts No Marks	15.0 pts
Evaluate and display the result of postfix expression. Verify that result is generated by postfix evaluation, and NOT infix evaluation.	15.0 pts Full Marks	0.0 pts No Marks	15.0 pts
Use of apt data structures - infix queue, postfix queue, operator stack (STL queue/stack usage is allowed)	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
Program loops till user enters "quit" - Keep prompting user for and evaluating multiple expressions until user decides to quit.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
Support the input infix expression domain: non-negative numbers. -2 points if decimal numbers not supported.	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
Support the output result domain: real numbers. Output may be a fraction. It may also be negative. (Output may be rounded to 2 digits or more. -4 points if output is rounded to whole numbers)	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
Support for operators {+, -, *, /, %} and parentheses (2 points per operator and 5 points for parentheses).	15.0 pts Full Marks	0.0 pts No Marks	15.0 pts
Evaluation follows BODMAS operator precedence (15 points for precedence logic).	15.0 pts Full Marks	0.0 pts No Marks	15.0 pts
[(+bonus points+)] Support for POW operator	10.0 pts Full Marks	0.0 pts No Marks	10.0 pts
Handle spaces in input infix expression by ignoring them.	3.0 pts Full Marks	0.0 pts No Marks	3.0 pts

Criteria	Ratings		Pts
Output postfix expression must be readable: operands and operators separated by 0 or more spaces, operands and operands separated by 1 or more spaces.	3.0 pts Full Marks	0.0 pts No Marks	3.0 pts
Error catching: Catch divide-by-zero exception, NaN exception. (Input infix expression can be assumed to be valid)	4.0 pts Full Marks	0.0 pts No Marks	4.0 pts
Miscellaneous Test Cases and Coding Style	5.0 pts Full Marks	0.0 pts No Marks	5.0 pts
Note: 30 points penalty for late submission!	0.0 pts Full Marks	0.0 pts No Marks	0.0 pts
			Total Points: 110.0