FLORIDA POLYTECHNIC UNIVERSITY

Points: 120/100

Due Date: Nov. 8th, 2021

Project #2: Reverse Polish Notation

Submission Notes:

1. In addition to the code, please upload your report submission in PDF format. The report should have a clean and clear run of the system, including screenshots.

2. Do not submit any zip or compressed files, binary files, or any other generated files. Do not put the code inside the PDF file. Submission of unnecessary files or binary files in addition to source files will make you lose the points of the assignment. The code must have the following:
    a. Use of descriptive names for the identifiers and active names for functions.
    b. Use consistent indentation through your code.
    c. Use consistent braces style through your code.
    d. Efficient utilization of functions. Make sure to reduce (*keep functions as simple as you can make them*) and reuse (*create functions for repeated parts of the code*) your code.
    e. File-level comments, these comments will include the file name, your name, section number, and date of last modification. In addition, these comments must have instructions on how we run the program and test it.
    f. Function level comments include the function's description, the function's parameters, and the function's return value. These comments will appear before each of the functions, not the prototypes.
    g. In function comments, these comments will include a description of the atomic functionalities within the bodies of the functions.

Project #2: Reverse Polish Notation

What is reverse polish notation?

The Reverse Polish notation (RPN), aka postfix, is a technique that is mainly used by the computer to parse and evaluate mathematical expressions by utilizing the stack. The regular mathematical notation that we use every day is called the infix notation.

Understanding the precedence of operations, i.e., Parentheses, Exponents, Multiplication/Division, Addition/Subtraction, makes it easy to convert an infix expression to RPN.

Example 1:

The RPN for

```
(50.1 + 30 ) / 2
```

is

```
50.1 30 + 2 /
```

Note that you can have an intermediate state by taking the reverse polish for every two operands:

```
(50.1 30 +) 2 /
```

Using stack and recursion to evaluate:

```
evaluate()
      operation   /
      operand2    2
      operand1    evaluate()
                        operation   +
                        operand2    30
                        operand1    50.1
                  return 80.1
return 40.05
```

Example 2:

The RPN for

```
400 + 40 * 100 + 30 - 60 / 4
```

is

```
400 40 100 * + 30 + 60 4 / -
```

Using stack and recursion to evaluate:

```
evaluate()
      operation   -
      operand2    evaluate()
                        operation   /
                        operand2    4
                        operand1    60
                  return 15
      operand1    evaluate
                        operation   +
                        operand2    30
                        operand1    evaluate()
                                          operation   +
                                          operand2    evaluate
                                                            operation   *
                                                            operand2    100
                                                            operand1    40
                                                      return 4000
                                          operand1    400
                                    return 4400
                  return 4430
return 4415
```

Project Description

Write a C++ program that utilizes the stack to perform mathematical calculations using the Reverse Polish Notation (RPN), also known as postfix notation.

1. Write your stack implementation that has the following functionalities:

   - a flexible capacity that can be set by the function `setCapacity(size_t capacity)`.
   - `push`, `pop`, and `top` functions.
   - `getSize()` function to get the number of elements on the stack.
   - `evaluate()` function is a recursive function that evaluates the expression in the stack, consumes the elements of the stack, and returns the result as `double`.

2. The stack elements should be able to hold both the operand (value) and the mathematical operator as a character.

3. The expression should be passed to your program as space-separated arguments. For example, assume your program executable is named `reversePolish.exe`, then you should be able to run your program as:

   ```
   reversePolish.exe 2700 + 80 2 /
   ```

   and it will display parse the expression into the stack, perform the evaluation, and print the result on the screen as:

   ```
   1390
   ```

4. The operations that your calculator should handle are `+`, `-`, `*`, and `/`.

Bonus

You can get a 20 points bonus by submitting another program that converts the standard (infix) notation to RPN and then using your above implementation to perform the calculation.

Guidelines

1. Make sure to include sufficient documentation for your code. That includes comments and choosing the appropriate names for your data members, functions, classes, and objects.
2. Proper formatting of the code.
3. Separate the implementation and interface in all classes.