CMPSC 122 section 2 **Homework 5** Due: Tuesday, April 26, 2022

Calculator

Reserve Polish Notation (RPN) is a form of mathematical expression that is written in a postfix notation. The following expression in infix notation

$$A + (B + C) * (D - E) * F + G / H;$$

can be rewritten as postfix notation as follows

$$ABC + DE - F * GH / + * +$$

without losing correctness and accuracy. Converting infix notation to postfix notation is easily done using a stack. Even though infix notation is easier to read for humans, the postfix notation has several advantages over infix notation: 1) it does not need parenthesis, 2) it is easy to evaluate its value using a stack, and 3) it leads to faster calculation.

The goal of this assignment is to implement two functions using stacks: one is to convert the infix notation to the postfix notation, and the other is to evaluate the postfix notation expression. To this end, you should implement series of classes and the two functions, as follows:

- `List` class: a doubly linked list template class;
- `Stack` class: a stack template class that uses `List` class;
- `InfixToPostfix()`: a function that converts infix notation to postfix notation;
- `EvaluatePostfix()`: a function that evaluates the postfix notation expression.

You are provided the following startup files:

```
list.h
stack.h
calculatortest.cpp
calculatorimpl.cpp
```

You can find the main function in `calculatortest.cpp`. You can compile and run the program using the following command:

```
$ g++ -ansi –pedantic -Wall calculatortest.cpp calculatorimpl.cpp –o calculator $
                                                                    ./calculator
```

The `calculator` program runs 32 test cases for `List` class, `Stack` class, `InfixToPostfix()` function, and `EvaluatePostfix()` function. You should implement missing and incomplete functions and member functions, so that all test cases are passed. If you pass all tests successfully, you will find the following messages:

```
Congratulation!
You successfully passed all tests!
```

Followings describe what you should implement.

1. (15 points) doubly linked lists

You are provided the header file of the template doubly linked list class:

`list.h`

Implement missing member functions in `list.h`. You can add additional member function(s) at the end of the classes if you need.

2. (15 points) stacks using the doubly linked lists

You are provided the header file of the stack that uses the template doubly linked list class:

`stack.h`

Implement its missing member functions in `stack.h`. You can add additional member function(s) at the end of the classes if you need.

3. (30 points) infix to postfix

You are provided the `InfixToPostfix()` function that converts infix expression to postfix expression in

`calculatorimpl.cpp`.

Its prototype is the same to the following: `string`

`InfixToPostfix(string infix)`.

The function receives one parameter `infix` that is an expression string in the form of infix notation, and should return its postfix notation (or RPN) expression. The string `infix` ends with an *ending delimiter* '`;`' and has two types of character:

- a single character *operand* that is one of the element of {'`0`', '`1`', '`2`', …, '`8`', '`9`'}, •
  a single character *operator* that is one of the element of {'`(`', '`)`', '`+`', '`-`', '`*`', '`/`' }.

For example, if the input is "`(1+3)/8;`" then its output should be "`13+8/`". Complete the implementation of `InfixToPostfix()` function in `calculatorimpl.cpp`.

4. (30 points) evaluate postfix

You are provided the `EvaluatePostfix()` function that evaluates the postfix expression in

`calculatorimpl.cpp`.

Its prototype is the same to the following: `double`

`EvaluatePostfix(string postfix)`.

The function receives one parameter `postfix` that is an expression string in the form of the postfix notation (or RPN) expression, and should return its calculated result in double. For example, if the input is "`13+8/`" then it should return `0.5` in double. Complete the implementation of `EvaluatePostfix()` function in `calculatorimpl.cpp.`

5. (10 points) memory management and coding style

You will receive 10 points from dynamic memory allocation and your coding style such as creating and deleting `Node` objects, indentation, variable names, and comments.

What to submit

- Electronic submission is due April 26 at midnight. Please submit a zip file containing the two header files (**list.h** and **stack.h**) and **calculatorimpl.cpp**. Do not submit calculatortest.cpp.