CISC2200 Spring 2023
Final Project 100 points **Due Date: Thursday, May 11, 2023**

In this project, you will implement a calculator which can do four basic calculations. The input of this calculator should be in the format of **infix notation**:

(6 + 13) * 2 – (5 + 1) / 3

To evaluate an infix expression, the calculator first converts the expression to **postfix notation**, and then evaluates it. The format of postfix notation of the above input is:

6 13 + 2 * 5 1 + 3 / -

The following arithmetic binary operations are allowed in an expression:

Binary operation	Infix Notation	Postfix Notation
+, -, *, /	***Operand1*** **Operator** ***Operand2***	***Operand1 Operand2*** **Operator**

The program should read the infix input from the standard input, print the expression in the postfix format on the screen, and then print the evaluated value of the expression.

```
Please type the expression in the format of Infix Notation :
(6 + 13) * 2 – (5 + 1) / 3
The postfix notation :
6 13  +  2 * 5  1  + 3  /  -
The result : 36
```

You should check errors of the input string in case it is not syntactically and/or semantically correct. In other words, there may be input such as "1+2*", “(1+2))” or "1/0". You should create your own **Exception classes** to handle them.

Algorithm to convert from infix to postfix (with stack):

1. Scan the given input from left to right,
2. For the next element in the input.
    a. If it is an operand, output it.
    b. If it is opening parenthesis, push it on the stack.
    c. If it is an operator, then
        i. If the stack is empty, push the operator on the stack.
        ii. If the top of the stack is opening parenthesis, push the operator on the stack.
        iii. If it has higher priority than the top of stack, push operator on stack.
        iv. Else pop the operator from the stack and output it, repeat step c.
    d. If it is a closing parenthesis, pop operators from the stack and output them until an opening parenthesis is encountered. Pop and discard the opening parenthesis.
3. If there are more inputs, go to step 2, else unstack the remaining operators to output.

Algorithm to evaluate postfix expression (with stack):

1. Scan the given postfix expression from left to right
2. for each token in the input postfix expression
    a. if the token is an operand
        i. push it (its value) onto the stack
    b. else if the token is an operator
        i. operand2 = the element popped from the top of the stack
        ii. operand1 = the element popped from the top of the stack
        iii. compute operand1 operator operand2
        iv. push result onto the stack
3. return top of the stack as result

You should use the <stack> template class from STL of C++ to finish this project.

Submit: email **yli@fordham.edu** your files: YourName_FinalProject.cpp