Assignment 5 – Infix Expression Calculators

Due Date: 11:59 pm October 24, Friday

Objectives:

- Students will gain an experience to evaluate arithmetic expressions by stacks
- Students will gain an experience to handle errors using C++ exceptions

Problem

Task1 – Exercises

Given the following expressions,

- `a-(b/c*d)`
- `a/b/c-(d+e)*f`
- `a*(b/c/d)+e`

use the algorithm learnt in Chapter 6 to convert these expression into the equivalent postfix expression. Illustrate the algorithm step by step in the following table:

ch	operatorStack (bottom to top)	postfixExp

Task2

Consider simple infix expressions that consist of single-digit operands; the operators +, -, *, and /; and the parentheses. Assume that unary operators are illegal and that the expression contains no embedded spaces. Design and implement a class of infix calculators. The class should have the following members:

Private members:

1. a string to store an infix expression for evaluation.
2. a private member function that checks if an infix expression is well formed.
3. a private member function that checks for balanced brackets of an expression using the algorithm studied in Chapter 8.
4. a private member function that converts an infix expression to a postfix expression using the algorithm studied in Chapter 8.
5. a private member function that determines the precedence of an operator
6. a private member function that evaluates a postfix expression using the algorithm studied in Chapter 8.

Public members:

1. a default constructor.
2. a function that sets the data member by an infix expression. The function must first check if the infix expression is well formed and balanced with brackets by calling private member function 2 and 3 before passing it to the data member. The function returns true if the operation is successfully performed. Otherwise it returns false to indicate the parameter expression is not valid.
3. a function that evaluates the expression stored in the data member. This function should not have any parameters. The function should first convert the infix expression to its postfix form and then evaluate the resulting postfix expression.

Other requirements

- Use the link-based Stack with exceptions.
- Write a main function to test your calculators. Your program should allow the user to evaluate additional expressions until the user wants to end the program.
- You should not assume that the expression is well-formed. The user is asked to reenter an expression if the expression entered is not well formed.
- You should not assume that the parentheses in the expression are balanced. The user is asked to reenter an expression if the expression entered is not balanced.

What to Hand In

- For Task1
  - No submission is needed but you must do it because I may ask a similar question in Test2. You can verify your answers by your program.
- For Task2
  - Submit all source programs to your class account in **GitHub** system.
  - Submit the following documents to the drop box **Project5** on D2L:
    - all source programs
    - a script file for test runs on **GitHub**
    - a word file that contains
      - design document for your infix calculators
      - test data with an explanation
      - a user document

    (Please follow the details of the document About Programming Assignments posted on D2L.)

Grading

Requirements	**points**
Comments in the program	10
Program correctness	40
Script file from several test runs on GitHub	20
Design document	10
Test data with explanation	10
User document	10
TOTAL POINTS	**100**