# Infix to Postfix Converter Project

## Project Objectives

* To encapsulate the converter operation inside a class.
* To use a local stack inside the converter member function.
* To include a separate member function that assigns a precedence value to an operator.
* To ensure that the operators are restricted to +, -, \*, /.
* To include pre-condition and post-condition comments in the class definition file.
* To ensure that input file processing is not included in the converter class (i.e., the input file is processed in the client program).
* To answer the reflection questions in the report that is posted to Canvas.

## Report Instructions

* Read the problem requirements in the project instructions section below.
* Plan the project.
* Go to the class team on replit and write and test your code in the designated project folder.
* When your project is ready to be graded write a project report with the following format:
* Name of program (title, number, or both)
* Your name
* Name of your partner (if you have one)
* Planning notes – UML diagrams, IPO charts for member functions, handwritten notes, etc.
* Answer the following reflection questions:
  + 1. What did you find most challenging with this program?
    2. What problems did you encounter and how did you solve them?
    3. What did you learn from writing this program?

Your project report must be in .pdf format. Upload your project report to the assignment in Canvas and submit.

## Project Instructions

Construct a class that converts an infix expression into an equivalent postfix express and a client program to test the class. Your class must utilize a stack structure as describe in the algorithm below. Use the **<stack>** class from the Standard Template Library (STL). There is only one instance variable for this class, the string to hold the postfix expression. Your class will need to provide an accessor function to allow the client to get the postfix expression after the conversion is complete. You can add member functions as needed to perform the conversion.

**Convert()**

**Precondition – An infix expression is provided; the infix expression is assumed to be well-formed**

**Postcondition – the infix expression has been converted to a postfix expression and stored in the instance variable**

**Algorithm -**

**Create a stack for the operators**

**Clear the postfix string**

**For each character in the input string**

**If the character is an alpha character**

**Append the character to the postfix string**

**Else if the character is a ‘(‘**

**Push the character onto the stack**

**Else if the character is ‘)’**

**If the stack is not empty**

**While the top of the stack is not a ‘(’**

**Pop the operator and append it to the postfix string**

**Pop the ‘(‘**

**Else if the character is a ‘ ’**

**continue to the next character**

**Else**

**while (the stack is not empty and**

**string operator precedence <= stack operator precedence**

**Pop and append the stack operator to the postfix string**

**Push the string operator onto the stack**

**End For**

**while the stack is not empty**

**pop and append remaining operators to the postfix string**

In this program, you will consider the following (binary) arithmetic operators: +, -, \*, and /.

A data file called InfixData.txt is provided which contains the following expressions:

**A + B – C**

**(A + B) \* C  
(A + B) \* (C - D)  
A + ((B + C) \* (E - F) - G) / (H - I)  
A + B \* (C + D) - E / F \* G + H**

Your output should look like this:

**Infix Expression: A+B-C;  
Postfix Expression: AB+C-  
  
Infix Expression: (A+B)\*C;  
Postfix Expression: AB+C\*  
  
Infix Expression: (A+B)\*(C-D);  
Postfix Expression: AB+CD-\*  
  
Infix Expression: A+((B+C)\*(E-F)-G)/(H-I);  
Postfix Expression: ABC+EF-\*G-HI-/+  
  
Infix Expression: A+B\*(C+D)-E/F\*G+H;  
Postfix Expression: ABCD+\*+EF/G\*-H+**