**Colorado Mesa University**

**Computer Science and Engineering**

**CSCI112**

**Practicum 12 / Assignment 5 Due November 15**

**Aim:**

To get some experience with STL, Try-Catch, and Assert

**Task:**

Redesign the code solution attached to the problem described below so that:

Task 1-A:

1. All dynamic memory allocations are done within try catch blocks, each throwing a unique error code and each having its own custom catcher + 1 only catch all.
2. Each catch block should attempt to resolve the issue if possible or rethrow to the catch all after writing to “CERR”. Redirect “CERR” to an error log file named run\_log.txt. Each error in the log should have at least information as to which try block generated it, what was the error, and time stamp.
3. Store all files and makefile in a folder labeled Task1A

Task 1-B:

1. Once you have completed and thoroughly tested Task 1-A, rewrite the code so that it uses the STL class **stack** to convert the infix expressions to postfix expressions.
2. Store only the required files for this task and its makefile in a folder labeled Task1B

**Problem (Infix to Postfix)**

This is a program that converts an infix expression into an equivalent postfix expression.

The rules to convert an infix expression into an equivalent postfix expression are as follows:

Suppose infx represents the infix expression and pfx represents the postfix expression. The rules to convert infx into pfx are as follows:

a. Initialize pfx to an empty expression and also initialize the stack.

b. Get the next symbol, sym, from infx.

b. 1. If sym is an operand, append sym to pfx.

b. 2. If sym is (, push sym into the stack.

b. 3. If sym is ), pop and append all of the symbols from the stack until the most recent left parentheses. Pop and discard the left parentheses.

b. 4. If sym is an operator:

b. 4.1. Pop and append all of the operators from the stack to pfx that are above the most recent left

parentheses and have precedence greater than or equal to sym.

b. 4.2. Push sym onto the stack.

c. After processing infx, some operators might be left in the stack.

Pop and append to pfx everything from the stack.

In this program, you will consider the following (binary) arithmetic operators:

+, -, \*, and /. You may assume that the expressions you will process are error free.

Design a class that stores the infix and postfix strings. The class must include the following operations:

1. getInfix: Stores the infix expression.

2. showInfix: Outputs the infix expression.

3. showPostfix: Outputs the postfix expression.

Some other operations that you might need are as follows:

1. convertToPostfix: Converts the infix expression into a postfix expression. The resulting postfix expression is stored in pfx.

2. precedence: Determines the precedence between two operators. If the first operator is of higher or equal precedence than the second operator, it returns the value true; otherwise, it returns the value false.

Include the constructors and destructors for automatic initialization and dynamic memory deallocation.

Test your program on the following expressions:

a. A + B - C;

b. (A + B ) \* C;

c. (A + B) \* (C - D);

d. A + ((B + C) \* (E - F) - G) / (H - I);

e. A + B \* (C + D) - E / F \* G + H;

For each expression, your answer must be in the following form:

Infix Expression: A + B - C;

Postfix Expression: A B + C

Put both folders Task1A and Task1B in a folder named STL Assignment, compress and submit by the due date.

Grading will be by demo – in person.