Assignment 3 : Implementing and Using the Stack ADT

Due Date: March 13 Monday 9:00am

Goal: In this project, you will implement the ***Stack ADT*** using the ***STL List class***. You will then use your implementation of Stack ADT to write a program that converts infix notations to postfix notations.

The following files are given to you:

- ***lstack.h***: Skeleton file for list-based stack implementation.
- ***i2p***: Sample executable for converting infix expressions to postfix.
- ***Grading guidelines.***

Deliverables: Please turn in a `makefile` and all header (*.h) and cpp (*.cpp) files that are needed to build your software. Turn in your development log too, which should be a plain ASCII text file called `LOG.txt` in your project directory. You will submit all of these using the campus blackboard submission system. Detailed submission instructions can be found by clicking on the **[Submissions]** link on the course website http://www.cs.fsu.edu/~kartik/cop4530/index.html

Background: Your first task is to write an ***adaptor class*** that implements the Stack ADT using the STL List class as the adaptee.

You are given the function interface for a class template called **LStack** in the file lstack.h:

```
template <typename Object>
class LStack {

    public:

        LStack( ); // constructor
        LStack(const LStack & rhs); // The copy constructor
        stack & operator=(const stack &); // assignment operator
        bool empty( ) const; // Returns true if the stack contains no elements,
        and false otherwise. S.empty( ) is equivalent to S.size( ) == 0.
        int size( ) const; // Returns the number of elements contained in the
        stack.
        Object & top( ); // Returns a mutable reference to the element at the top
        of the stack. Precondition: empty( ) is false.
        const Object & top( ) const; //Returns a const reference to the element
        at the top of the stack. Precondition: empty( ) is false.
        void push(const Object & x); // Inserts x at the top of the stack.
        Postconditions: size( ) will be incremented by 1, and top( ) will be equal
        to x.
        void pop( ); // Removes the element at the top of the stack.
        Precondition: empty( ) is false. Postcondition: size( ) will be
        decremented by 1.

    private:

        list<Object> store; // STL List storing the objects in the stack
```

// **Note 1 :** You should add any additional private or protected members that you need.
// **Note 2 :** You don't need any iterators for the stack. Why?

}

Requirements:

1. You are asked to complete the implementation of the LStack class template given above. Your LStack implementation should support the same functional interface and behaviour as the STL stack class.
2. Write a program, called **i2p**, that uses your LStack implementation to *convert* an arithmetic expressions from **Infix** format to **Postfix** format.
   - Your program should read the infix format expression from the standard input and write its postfix format expression to the standard output.
   - Assume that operands are represented by single characters 'a', 'b', 'c' etc. Also assume that there is no space between any symbols in the input. For example *a+b+c+d*.
   - The infix expression can include any of the following operators: (,) , /, *, + , - . The conversion should take into account the precedence ordering among the operators.
   - The process of conversion is explained in section 3.6.3 in the book. For your reference, here are some elementary examples for conversion from infix to postfix.

a+(b+c)	→	abc++
(a+b)+c	→	ab+c+
a-b*c	→	abc*-
(a/b)*(c/d)	→	ab/cd/*
a/(b+c*d-e)	→	abcd*+e-/
a-b*c+d/e	→	abc*-de/+

Thus an infix expression *a + b*(c - d)+ e* can be written as *abcd - * + e +*. Notice that the translation of expressions that are not fully bracketed can appear ambiguous e.g. *a + b + c* could be translated to either *abc++* or *ab+c+*. The right choice is *ab+c+*. Why? Because + and - operators associate left to right. To understand this, think about how you will translate *a-b+c*.

   - You do not have to evaluate the expression. Just convert from infix to postfix format.
   - A sample executable program for i2p can be downloaded from here . Download and save the executable file to linprog machine, give execute permissions using **chmod +x i2p**, and execute the program.

- Feel free to add any necessary member functions or fields in your implementation to achieve the above goals.
- ***However, you should not use any STL class other than the STL list class in your implementation.***

Bonus points (10):

The **Towers of Hanoi** is a classic puzzle that combines the concepts of stacks, recursion and

recurrence relations. Write a program to solve the **Towers of Hanoi** puzzle in minimum number of steps. The description of the puzzle is given at the following website. http://mathworld.wolfram.com/TowerofHanoi.html Your program should take the number of disks in the tower as a command-line parameter. To obtain credit for your solution, please demonstrate the program in person to either the instructor or the TA.