DSA Summer 2014 – Lab 03

Task 1

You must be familiar with infix, postfix and prefix expressions. An infix expression is one in which operators are written infix-style between the operands they act on. For example, a+b*c+d.

A postfix expression is one in which operators are written postfix-style after the operands they act on. For example, abc*d++.

a) You have to implement a function **infix_to_postfix (string exp)** which takes an infix expression as input and returns an equivalent postfix expression. Use your own stack ADT which you made in the Lab pre-requisite task.
For those who don't know the use of "string" object can use their own CString object or they can simply use character pointer.

b) A driver function that takes an expression as input from user, passes the expression to both functions and shows the desired result on console.

Task 2

As now, you are familiar with converting infix to postfix and prefix expressions. So you must program to evaluate the converted expressions.

a) A function **evaluate_postfix1(exp)** which takes a **postfix** expression as input, evaluates it and returns the desired result. Each operand and operation symbol is separated by a single space.
Example,
Input: 5 9 3 / 2 * + 7 –
Output: 4

b) A driver function that takes prefix and postfix expressions from user as input, pass the expressions to the relevant function and shows the desired result on the console.

c) A function **evaluate_postfix2(exp)** which takes a **postfix** expression as input, evaluates it and returns the desired result. Each operand and operation symbol is separated by a single comma.
You may use string tokenizer.
Example,
Input: 5 9, 3, /, 21, *, 31,/,7, –   Output: Compute on your own.

Note: You have to check the validity of expressions too.