CL-2001
Data Structures
Lab # 6

Objectives:

- Stack ADT
- Stack Linked list
- Infix
- Prefix
- Postfix

Note: Carefully read the following instructions (*Each instruction contains a weightage*)

1. There must be a block of comments at start of every question's code by students; the block should contain brief description about functionality of code.
2. Comment on every function and about its functionality.
3. Mention comments where necessary such as comments with variables, loop, classes etc to increase code understandability.
4. Use understandable name of variables.
5. Proper indentation of code is essential.
6. Write a code in C++ language.
7. Make a Microsoft Word file and paste all of your C++ code with all possible screenshots of every task **outputs in Microsoft Word and submit word file. Do not submit .cpp file.**
8. First think about statement problems and then write/draw your logic on copy.
9. After copy pencil work, code the problem statement on MS Studio C++ compiler.
10. At the end when you done your tasks, attached C++ created files in MS word file and make your submission on Google Classroom. (Make sure your submission is completed).
11. Please submit your file in this format **19F1234_L4**.
12. Do not submit your assignment after deadline. Late and email submission is not accepted.
13. Do not copy code from any source otherwise you will be penalized with negative marks.

Problem: 1 | Stack Linked List

Stacks are used by compilers to help in the process of evaluating expressions and generating machine language code. In this and the next exercise, we investigate how compilers evaluate arithmetic expressions consisting only of constants, operators and parentheses.

Humans generally write expressions like 3 + 4 and 7 / 9 in which the operator (+ or / here) is written between its operands—this is called infix notation. Computers "prefer" postfix notation in which the operator is written to the right of its two operands. The preceding infix expressions would appear in postfix notation as 3 4 + and 7 9 /, respectively.

To evaluate a complex infix expression, a compiler would first convert the expression to postfix notation and evaluate the postfix version of the expression. Each of these algorithms requires\ only a single left-to-right pass of the expression. Each algorithm uses a stack object in support of its operation, and in each algorithm the stack is used for a different purpose.

In this exercise, you'll write a C++ version of the infix-to-postfix conversion algorithm. In the next exercise, you'll write a C++ version of the postfix expression evaluation algorithm. Later in the chapter, you'll discover that code you write in this exercise can help you implement a complete working compiler

Write a program that converts an ordinary infix arithmetic expression (assume a valid expression is entered) with single-digit integers such as

(6 + 2) * 5 - 8 / 4

to a postfix expression. The postfix version of the preceding infix expression is

6 2 + 5 * 8 4 / -

The program should read the expression into string infix and use modified versions of the stack functions implemented in this chapter to help create the postfix expression in string postfix. The algorithm for creating a postfix expression is as follows:

Push a left parenthesis '(' onto the stack. 2) Append a right parenthesis ')' to the end of infix. 3) While the stack is not empty, read infix from left to right and do the following:

If the current character in infix is a digit, copy it to the next element of postfix. If the current character in infix is a left parenthesis, push it onto the stack. If the current character in infix is an operator, Pop operators (if there are any) at the top of the stack while they have equal or higher precedence than the current operator, and insert the popped operators in postfix.

Push the current character in infix onto the stack. If the current character in infix is a right parenthesis Pop operators from the top of the stack and insert them in postfix until a left parenthesis is at the top of the stack. Pop (and discard) the left parenthesis from the stack.

The following arithmetic operations are allowed in an expression:

+ addition

- subtraction

* multiplication

/ division

^ exponentiation

[**Note**: We assume left-to-right associativity for all operators for the purpose of this exercise.] The stack should be maintained with stack nodes, each containing a data member and a pointer to the next stack node.

Some of the functional capabilities you may want to provide are:

a) function convertToPostfix that converts the infix expression to postfix notation

b) function precedence that determines whether the precedence of operator1 is greater than or equalt to the precedence of operator2, and, if so, returns true.

c) function push that pushes a value onto the stack

d) function pop that pops a value off the stack

e) function stackTop that returns the top value of the stack without popping the stack

f) function isEmpty that determines if the stack is empty

Driver Code:

```
#include <iostream>
#include <conio.h>
#include <string.h>
using namespace std;
struct node
{
	int data;
	node *next;
}*p = NULL, *top = NULL, *save = NULL, *ptr;
void push(int x){}
```

```
char pop(){}
bool isEmpty{}
int precedence(char oper){}
string convertToPostfix (string pre){}
}
int main()
{
	string pre;
	cout << "enter the balanced expression\n";
	cin >> pre;
	cout << "Postfix : " << convertToPostfix(pre);
	system("pause");
}
```

Use the problem code to evaluate the expression.

"a+b*(c^d-e)^(f+g*h)-I" (This notation will be passed to the program as a string)

Problem: 2 | Stack Linked List

Write a code to change the following postfix notation to Infix notation. Check for the following input.

abc++

Problem: 3 | Stack Linked List

Also write a C++ program to convert the following prefix notation to postfix notation. Run the code on the following input.

*+AB-CD

☺ Best of luck