East West University
Department of Computer Science & Engineering
A/2, Jahurul Islam Avenue, Jahurul Islam City, Aftabnagar, Dhaka-1212

Lab Manual **:** 03
Course Code : CSE207
Course Title : Data Structure
Instructor : Tanni Mittra, Lecturer, CSE

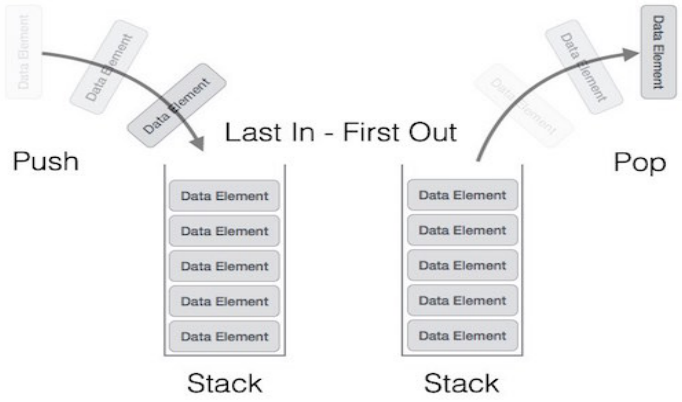
Objective:

The objective of this lab is to provide basic concept of stack. At the end of the lab, students are able:

- To learn how to create a stack
- To learn how to perform push and pop operation in stack
- To learn how to use stack for parsing unmatched parenthesis in an algebraic expression
- To learn how to use stack for reversing data.

Stack:

A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc. A real-world stack allows operations at one end only. This feature makes it LIFO data structure. In stack terminology, insertion operation is called PUSH operation and removal operation is called POP operation.



Now you have to perform the following lab task on stack. At first build a stack ADT. Use the way that I have taught you in class. Then use the stack ADT in a new program file and implement the exercises mentioned in the lab manual.

Exercise 1:

Write the program called copyStack that copies the contents of one stack into another. The algorithm passes two stacks, the source stack and the destination stack. The order of the stacks must be identical.

Exercise 2:

Reversing Data

Reversing data requires that a given set of data be reordered so that the first and last elements are exchanged. The idea of reversing data can be used in solving classical problem such as converting a decimal number to a binary number. Now write a program using stack that will convert decimal number to binary number. For example:

Input	**Output**
45	101101
4	100

Exercise 3:

Write a program that reverses the contents of a stack (the top and bottom elements exchange positions, the second and the element just before the bottom exchange positions, and so forth until the entire stack is reversed). (Hint: Use temporary stacks.)

Exercise 4:

Infix to postfix conversion

Write a program that implements the infix-to-postfix notation. The program should read an infix string consisting of single alphabetic characters for variables, parentheses, and the +, -, *, and / operators.

Exercise 5:

Parenthesis parsing

Write a program using stack that will parse parenthesis of an equation. Use stack data structure for parsing.