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

Assignment : **02**
Course Code : **CSE207**
Topic : **Stack**
Course Title : **Data Structures**
Instructor : **Md. Manowarul Islam, Adjunct Faculty, Department of CSE**

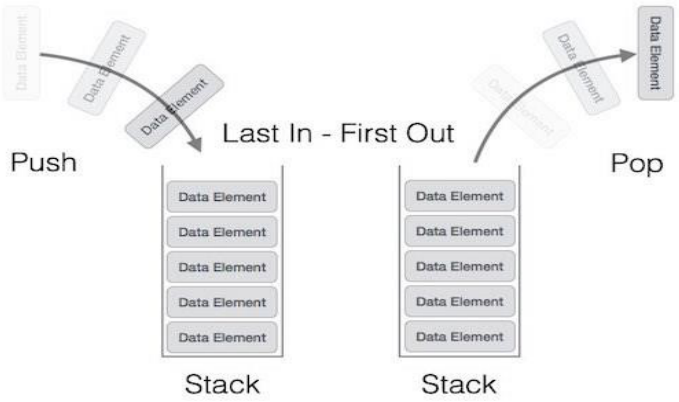
Objective:

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 must perform the following task on stack:

Exercise 1: (Stack using array)

Create a Menu

Create a menu that will display all the exercises given below (Exercise 2 to Exercise 5) as a list and prompt user to select any desired option. The menu can be designed in below format.

1. Insert data/ push stack
2. Pop stack
3. Peek stack
4. Print stack
5. Exit

Push Operation

Adding a new data in stack is a more than one step activity. First, create an array of size N. Then input the data and store it in the space using one variable TOP.

Pop Operation

After completing exercise push operation, you have a newly created stack. Now perform the pop operation on it.

Peek Operation

Display the top/ last inserted value of the stack.

Print the stack

After completing pop operation print the stack.

Exercise 2:

Parsing Unmatched Parenthesis

One of the most important applications of stack is parsing. Parsing is any logic that breaks data into independent piece for further processing. So, parsing unmatched parenthesis is a common problem of parsing. When parentheses are unmatched then there will be two types of error: the opening parenthesis is unmatched, or the closing parenthesis is missing.

Write a program using stack that will make sure that all parentheses are well paired. For example,

Input	Output
((A+B)/C	Opening parentheses do not end
(A+B)/C)	Closing parentheses not matched

Exercise 3:

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 4:

Sort a Stack

Given a stack of integers, your task is to sort the stack in **ascending** order using only stack operations (push, pop, peek). You are not allowed to use any other data structures like arrays.

Constraints:

- You can only use two stack operations: **push** and **pop**.
- The input stack may not be sorted initially.

Input	Output
Enter elements: 10, 3, 2, 1, 4, 7	Sorted stack: 1, 2, 3, 4, 7, 10

Exercise 5:

Find the Middle Element of a Stack

You are given a stack. Your task is to find the **middle element** of the stack. The middle element is defined as the element that is exactly in the middle of the stack, where the elements are arranged in a linear order. If the stack contains an even number of elements, return the element just below the middle.

Constraints:

- You can only use stack operations: **push**, **pop**, and **peek**.
- The stack is non-empty

Input	Output
Enter elements: 10, 20, 30, 40, 50	Middle Element: 30

Exercise 6:

Next Greater Element

Given an array of integers, for each element in the array, find the **next greater element**. The next greater element is the first element to the right of the current element that is larger than it. If there is no greater element to the right, return -1 for that element..

You need to implement this problem using a **stack**.

Input	Output
Enter elements: 4, 5, 2, 10, 8	5, 10, 10, -1, -1
Enter elements: 1, 3, 2, 4	3, 4, 4, -1