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

Lab Manual : **01**
Course Code : **CSE207**
Course Title : **Data Structures**
Instructor : **Md. Manowarul Islam, Adjunct Faculty, Department of CSE**

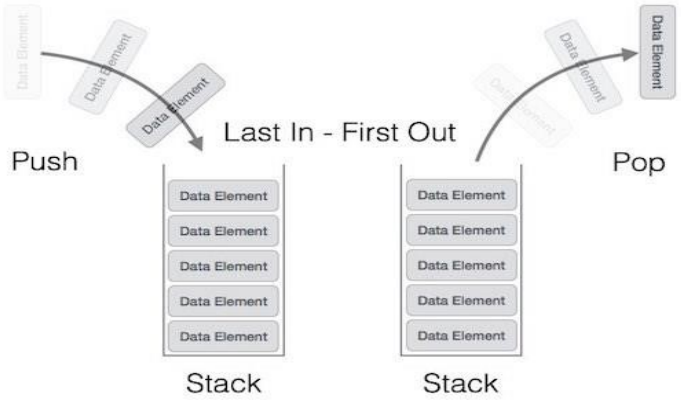
Objective:

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 have to perform the following lab 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. Print stack
3. Pop stack

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.

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 paried. For example,

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

Exercise 3:

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. *An input string is valid if:*

1. *Open brackets must be closed by the same type of brackets.*
2. *Open brackets must be closed in the correct order.*

The Pseudocode

Before we dive in, let's do a little pseudocode & see how our code might look in plain English:

1. We'll need to create our stack to hold our open parentheses'. This will start off as an empty array.
2. Set up our for loop, which will iterate through our input string
3. During each iteration, if our current element is an open parentheses ('**(**' or '**{**' or '**[**'), let's push that element into the top of our stack
4. During each iteration, if our current element is a closing bracket ('**)**' or '**}**' or '**]**'), let's pop off the last opening parentheses element from the stack ONLY if it matches with the encountered closing bracket
5. Remember, order matters here. If the closing bracket that we encounter does not match with the opening bracket placed on top of the stack, we then immediately break out of the loop and return `false` because the parentheses in the string are not balanced.
6. If the stack is empty after we've finished our iteration, we can conclude that our string contains a balanced parentheses and we can return `true`

Input		Output
{{}}()[()]	-->	The parentheses are valid
{][}	-->	The parentheses are invalid
)(	-->	The parentheses are invalid

Exercise 4:

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