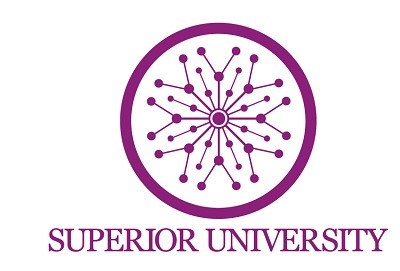
**Lab 10**



Submitted to

:

Sir Rasikh Ali

rR

Submitted by

:

Arooj Fatima

Roll No

:

SU92-BSSEM-S24-091

Subject:

DSA (Lab)

Class:

BS – Software Engineering

# **Lab # 10**

**Stack with LinkedList and Array Tasks:**

**A-With Array; Push, Pop, Display**

This code implements a stack data structure using an array (size 10) with standard **LIFO operations (push/pop)** and two special operations (**pushAtStart**/**popAtStart**). The top variable tracks the stack's current position, where push adds elements to the top (end of array) and pop removes them. The special operations work against typical stack behavior: **pushAtStart** inserts elements at the bottom **(index 0)** by shifting all elements right, while **popAtStart** removes the bottom element by shifting left. These unconventional operations are useful for scenarios requiring stack-like behavior with occasional access to the base, though they incur **O(n) time complexity** due to element shifting. The print method displays the stack top-to-bottom, and error messages prevent overflow/underflow. This implementation demonstrates both classic stack principles and flexible modifications for specific use cases.

**Output:**



**B-** **With LinkedList; Push, Pop, Display**

This code implements a stack using a linked list, where each Node contains data and a pointer to the next node. The standard **LIFO** operations (**push**/**pop**) modify the stack's top: push creates a new head node, while **pop** removes it. Two special operations (**pushAtStart**/**popAtStart**) manipulate the stack's tail (bottom) instead - **pushAtStart** appends a node at the end by traversing the list, while **popAtStart** removes the tail node by tracking the previous node. The print method displays the stack from top to bottom. Unlike array-based stacks, this dynamic implementation avoids size limits but requires pointer management. The special operations (**O(n) complexity**) enable rare tail access while maintaining core stack behavior, useful for hybrid stack/queue scenarios. Error handling prevents operations on empty stacks ("Underflow").

**Output:**

