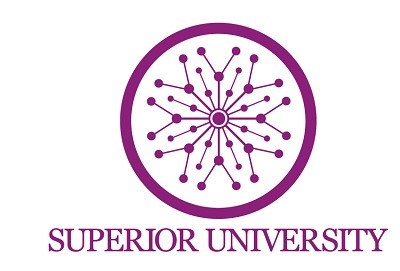
**Lab 11**



Submitted to

:

Sir Rasikh Ali

rR

Submitted by

:

Arooj Fatima

Roll No

:

SU92-BSSEM-S24-091

Subject:

DSA (Lab)

Class:

BS – Software Engineering

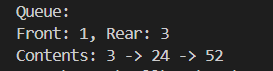
# **Lab # 11**

**Queue with LinkedList and Array Tasks:**

**A-** **With Array; Enqueue, Dequeue, Display**

This code implements a basic queue data structure using a fixed-size array (capacity 10) with **FIFO** (**First-In-First-Out**) operations. The **queue** tracks elements using **front** and **rear** indices, where **enqueue** adds elements at the rear (with checks for overflow) and **dequeue** removes elements from the front (with checks for underflow). The **print** method displays the queue contents along with the current **front/rear** positions. Unlike the stack implementations previously shown, this maintains strict **FIFO** ordering - elements are always added at the end and removed from the front. The array-based approach provides **O(1)** operations but has fixed capacity, and the implementation includes proper boundary checks to prevent queue underflow/overflow. The example demonstrates queue behavior by adding four numbers (6, 3, 24, 52) and then removing one element before printing.

**Output:**



**B-** **With Linkedlist; Enqueue, Dequeue, Display**

This code implements a queue data structure using a linked list approach with Node objects, where each node contains data and a pointer to the next node. The Queue class maintains two pointers - **front** (points to the first element) and **rear** (points to the last element) - to enable efficient FIFO (First-In-First-Out) operations. The **enqueue** operation adds new elements at the rear by creating a new node and updating the rear pointer, while **dequeue** removes elements from the front by updating the front pointer and freeing memory. The implementation automatically handles empty queue conditions (when front and rear are nullptr) and includes a **print** method that displays the queue elements from front to rear. Unlike the array-based version, this dynamic implementation has no fixed size limit and efficiently manages memory through node allocation/deallocation, though it requires slightly more memory for pointers. The example demonstrates queue operations by enqueuing four elements (1, 2, 3, 4) and then dequeuing one before printing the remaining elements (2 -> 3 -> 4).

**Output:**

