Experiment No.7
Implement Double Ended Queue using Linked List.
Date of Performance:
Date of Submission:

Experiment No. 7: Implement Double Ended Queue using Linked List.

Aim: Implementation of Double Ended Queue using Linked List.

Objectives:

1) Understand the Double Ended Queue (Deque) data structure and its ability to add or remove elements from both ends.
2) Learn to implement a Deque using a Linked List by creating a Node class and a Deque class.
3) Develop functions to perform operations like inserting at the front, inserting at the rear, deleting from the front, deleting from the rear, peeking at the front, and peeking at the rear.

Theory:

Deque (Double Ended Queue): A Deque is a linear data structure that allows insertion and deletion of elements from both ends—front and rear. Unlike a regular queue that follows First In First Out (FIFO), a Deque provides more flexibility by supporting operations at both ends.

Basic Operations on Deque

1. InsertFront(): Adds an element to the front of the deque.
2. InsertRear(): Adds an element to the rear of the deque.
3. DeleteFront(): Removes an element from the front of the deque.
4. DeleteRear(): Removes an element from the rear of the deque.
5. PeekFront(): Returns the element at the front of the deque without removing it.
6. PeekRear(): Returns the element at the rear of the deque without removing it.

Algorithm:

We maintain two pointers:

- front – points to the front of the deque
- rear – points to the rear of the deque

1. Insert at Front

- Create a new node
- Set newnode->next to front

- Update front->prev to new node (if exists)
- Update front to the new node
- If deque was empty, set rear = front = newnode

2. Insert at Rear

- Create a new node
- Set newnode->prev to rear
- Update rear->next to new node (if exists)
- Update rear to the new node
- If deque was empty, set front = rear = newnode

3. Delete from Front

- If front is not NULL:
  - Set temp = front
  - Move front to front->next
  - Update front->prev to NULL
  - free (temp)

4. Delete from Rear

- If rear is not NULL:
  - Set temp = rear
  - Move rear to rear->prev
  - Update rear->next to NULL
  - free (temp)

5. Peek Operations

- **Peek Front**: Return front->data
- **Peek Rear**: Return rear->data

6. isEmpty()

- Return true if both front == NULL and rear == NULL

Code and Output:

Conclusion: Students will conclude their understanding of Implementation of Double Ended Queue (DEQueue) using Linked List.