LAB SESSION 07

Queue Implementation Using Linked List

THEORY

A queue is a linear data structure that follows the First In First Out (FIFO) principle. In a linked list implementation of a queue, each element is represented as a node containing:

1. **Data** – the value stored in the node.
2. **Next pointer** – a reference to the next node in the queue.

For. Example
Front → [10] → [20] → [30] ← Rear
Here:

- **Front** is the first node (10), removed first when dequeued.
- **Rear** is the last node (30), where new elements are enqueued.

Key Pointers in Linked List Queue

- **front**: Points to the first node (element to be dequeued next).
- **rear**: Points to the last node (element most recently enqueued).

How It Works

- **Enqueue**: Create a new node and link it at the **end** (rear).
- **Dequeue**: Remove the node from the **front**.
- If front becomes NULL, the queue is empty.

Advantages Over Array Queue

- Dynamic size (no fixed maximum length).
- No wasted space due to shifting elements.
- Efficient insertion and deletion.

Disadvantages

- Requires extra memory for storing pointers.
- Slightly slower access compared to array due to pointer traversal.

PROCEDURE

1. Define a Node structure containing data and next.
2. Maintain front and rear pointers.
3. Implement:
4. enqueue() to insert at the rear.
5. dequeue() to remove from the front.
6. Display the queue after each operation.
7. Compile and run the program.

CODE

```
#include <iostream>
using namespace std;

struct Node {
```

```
    int data;
    Node* next;
};

Node* front = NULL;
Node* rear = NULL;

void enqueue(int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = NULL;
    if (rear == NULL) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    cout << value << " enqueued into queue.\n";
}

void dequeue() {
    if (front == NULL) {
        cout << "Queue Underflow!\n";
        return;
    }
    Node* temp = front;
    cout << temp->data << " dequeued from queue.\n";
    front = front->next;
    if (front == NULL) rear = NULL;
    delete temp;
}

void display() {
    if (front == NULL) {
        cout << "Queue is empty.\n";
        return;
    }
    cout << "Queue elements: ";
    Node* temp = front;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
```

```
    display();
    dequeue();
    display();
    return 0;
}
```

EXPECTED OUTPUT

```
10 enqueued into queue.
20 enqueued into queue.
30 enqueued into queue.
Queue elements: 10 20 30
10 dequeued from queue.
Queue elements: 20 30
```

EXERCISE

1. Modify the code to take user input for enqueue and dequeue.
2. Implement a **peek()** function to see the front element without removal.
3. Implement a circular queue using a linked list.
4. Count and display the total number of elements in the queue.
5. Write a function to clear all elements in the queue.