ASSISMENT 3

Student Name: Bhaskar Kumar	**UID: 23BCS14337**
Branch:BE-CSE	**Section/Group: KRG-3B**
Semester: 5	**Date of Performance:31-07-25**
Subject Name: DAA	
Subject Code:23CSH-301	

1. Aim: WAP of Queue using Array and LinkedList.

2. Objective: To develop a program showing insertion and deletion in a Stack (using C++) using array and linked list.

->Insertion(Enqueue) in Queue

->Deletion(Dequeue) in Queue

Queue follows FIFO(First In First Out)

- **Enqueue – Add to the queue**
- **Dequeue – Remove from queue**

3. CODE:

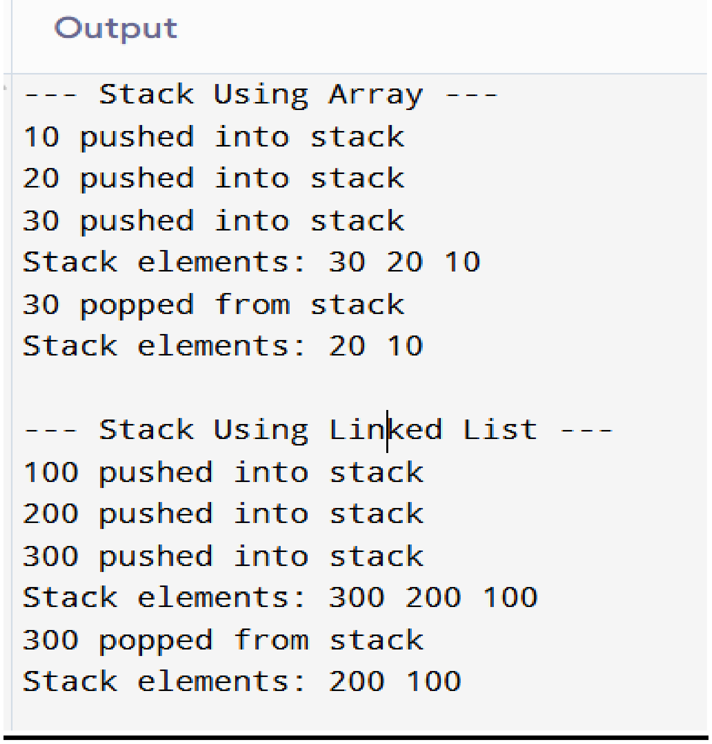
```
#include <bits/stdc++.h>
using namespace std;
//Using Array
class QueueArray {
   int front, rear, size, capacity;
   int* arr;
public:
   QueueArray(int n) {
      capacity = n;
```

```
    arr = new int[n];
    front = 0;
    rear = -1;
    size = 0;
}
void enqueue(int val) {
    if (size == capacity) {
        cout << "Queue Overflow!" << endl;
        return;
    }
    rear = (rear + 1) % capacity;  // circular increment
    arr[rear] = val;
    size++;
    cout << val << " enqueued into queue\n";
}
void dequeue() {
    if (size == 0) {
        cout << "Queue Underflow!" << endl;
        return;
    }
    cout << arr[front] << " dequeued from queue\n";
    front = (front + 1) % capacity;
    size--;
}
```

```
    void display() {
        if (size == 0) {
            cout << "Queue is empty!\n";
            return;
        }
        cout << "Queue elements: ";
        for (int i = 0; i < size; i++) {
            cout << arr[(front + i) % capacity] << " ";
        }
        cout << endl;
    }
};
//Using Linked List
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(NULL) {}
};
class QueueLinkedList {
    Node* front;
    Node* rear;
public:
    QueueLinkedList() {
        front = rear = NULL;
    }


    void enqueue(int val) {
        Node* newNode = new Node(val);
        if (!rear) {  // empty queue
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
        cout << val << " enqueued into queue\n";
    }


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

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

int main() {
    cout << "--- Queue Using Array ---\n";
    QueueArray q1(5);
    q1.enqueue(10);
    q1.enqueue(20);
    q1.enqueue(30);
    q1.display();
    q1.dequeue();
    q1.display();
    cout << "\n--- Queue Using Linked List ---\n";

    QueueLinkedList q2;

    q2.enqueue(100);

    q2.enqueue(200);

    q2.enqueue(300);

    q2.display();

    q2.dequeue();

    q2.display();


    return 0;

}
```

OUTPUT:



```
--- Stack Using Array ---
10 pushed into stack
20 pushed into stack
30 pushed into stack
Stack elements: 30 20 10
30 popped from stack
Stack elements: 20 10

--- Stack Using Linked List ---
100 pushed into stack
200 pushed into stack
300 pushed into stack
Stack elements: 300 200 100
300 popped from stack
Stack elements: 200 100
```

4.Learning Outcomes:

- Learned the concept and working of the **Queue data structure (FIFO)**.
- Implemented **enqueue and dequeue operations using both array and linked list** in C++.
- Understood **front, rear operations** and **overflow/underflow conditions** in queues.
- Analyzed the **time complexity and memory usage** of array vs linked list queue implementations.

5. Algorithm Analysis: