1. Queue Using Array

```
#include <iostream>
#define SIZE 5  // Define a constant SIZE for the array size

// Class representing a Queue using an array
class Queue {
private:
    int arr[SIZE];  // Declare an array to hold queue elements
    int front, rear;  // Declare two pointers, front and rear

public:
    // Constructor to initialize front and rear
    Queue() {
        front = -1;  // Start front at -1 to signify an empty queue
        rear = -1;   // Start rear at -1 to signify an empty queue
    }
```

```
    // Enqueue operation to add an element to the queue
    void enqueue(int value) {
        if (rear == SIZE - 1) {  // Check if the queue is full
            std::cout << "Queue is full\n";
        } else {
            if (front == -1) front = 0;  // If first element, set front to 0
            rear++;  // Move rear to next position
            arr[rear] = value;  // Insert value at rear
            std::cout << "Inserted " << value << std::endl;
        }
    }
```

```
    // Dequeue operation to remove an element from the queue
    void dequeue() {
        if (front == -1 || front > rear) {  // Check if the queue is empty
            std::cout << "Queue is empty\n";
        } else {
            std::cout << "Deleted " << arr[front] << std::endl;
            front++;  // Move front to next position
        }
    }
```

```
    // Display operation to show all elements in the queue
    void display() {
        if (front == -1 || front > rear) {  // Check if the queue is empty
            std::cout << "Queue is empty\n";
        } else {
            for (int i = front; i <= rear; i++) {  // Loop from front to rear
                std::cout << arr[i] << " ";  // Display each element
            }
            std::cout << std::endl;
        }
    }
};
```

```
int main() {
```

```
    Queue q;  // Create a Queue object
    q.enqueue(10);  // Add element 10 to the queue
    q.enqueue(20);  // Add element 20 to the queue
    q.enqueue(30);  // Add element 30 to the queue
    q.display();  // Display the queue
    q.dequeue();  // Remove an element from the queue
    q.display();  // Display the queue again
    return 0;
}
```

---

2. Queue Using Linked List

```
#include <iostream>

// Node class to create elements for the linked list queue
class Node {
public:
    int data;  // Data part of the node
    Node* next;  // Pointer to the next node
    Node(int val) : data(val), next(nullptr) {}  // Initialize node with
value
};

// Queue class using linked list
class Queue {
private:
    Node *front, *rear;  // Pointers to front and rear of the queue

public:
    // Constructor to initialize front and rear
    Queue() {
        front = nullptr;  // Start front as nullptr (empty queue)
        rear = nullptr;   // Start rear as nullptr (empty queue)
    }
```

```
    // Enqueue operation to add an element
    void enqueue(int value) {
        Node* newNode = new Node(value);  // Create a new node
        if (rear == nullptr) {  // If the queue is empty
            front = rear = newNode;  // Both front and rear point to new node
        } else {
            rear->next = newNode;  // Link new node after rear
            rear = newNode;  // Move rear to new node
        }
        std::cout << "Inserted " << value << std::endl;
    }
```

```
    // Dequeue operation to remove an element
    void dequeue() {
        if (front == nullptr) {  // Check if the queue is empty
            std::cout << "Queue is empty\n";
        } else {
```

```
            Node* temp = front;  // Temporary pointer to front node
            std::cout << "Deleted " << front->data << std::endl;
            front = front->next;  // Move front to the next node
            if (front == nullptr) rear = nullptr;  // If queue becomes empty
            delete temp;  // Delete the removed node
        }
    }
```

```
    // Display operation to show all elements in the queue
    void display() {
        if (front == nullptr) {  // Check if the queue is empty
            std::cout << "Queue is empty\n";
        } else {
            Node* temp = front;  // Temporary pointer to traverse the queue
            while (temp != nullptr) {  // Traverse till end of queue
                std::cout << temp->data << " ";  // Display node data
                temp = temp->next;  // Move to the next node
            }
            std::cout << std::endl;
        }
    }
};
```

```
int main() {
    Queue q;  // Create a Queue object
    q.enqueue(10);  // Add element 10 to the queue
    q.enqueue(20);  // Add element 20 to the queue
    q.enqueue(30);  // Add element 30 to the queue
    q.display();  // Display the queue
    q.dequeue();  // Remove an element from the queue
    q.display();  // Display the queue again
    return 0;
}
```

3. Queue Using Circular Array

```
#include <iostream>
#define SIZE 5  // Define a constant SIZE for the array size

// Class representing a Circular Queue using an array
class CircularQueue {
private:
    int arr[SIZE];  // Array to store queue elements
    int front, rear;  // Two pointers, front and rear

public:
    // Constructor to initialize front and rear
    CircularQueue() {
        front = -1;  // Start front at -1 to signify an empty queue
        rear = -1;   // Start rear at -1 to signify an empty queue
    }
```

```
    // Enqueue operation to add an element
    void enqueue(int value) {
        if ((rear + 1) % SIZE == front) {  // Check if the queue is full
            std::cout << "Queue is full\n";
        } else {
            if (front == -1) front = 0;  // Set front to 0 for first element
            rear = (rear + 1) % SIZE;  // Move rear circularly
            arr[rear] = value;  // Insert value at rear
            std::cout << "Inserted " << value << std::endl;
        }
    }
```

```
    // Dequeue operation to remove an element
    void dequeue() {
        if (front == -1) {  // Check if the queue is empty
            std::cout << "Queue is empty\n";
        } else {
            std::cout << "Deleted " << arr[front] << std::endl;
            if (front == rear) {  // If only one element was present
                front = -1;
                rear = -1;
            } else {
                front = (front + 1) % SIZE;  // Move front circularly
            }
        }
    }
```

```
    // Display operation to show elements in the queue
    void display() {
        if (front == -1) {  // Check if the queue is empty
            std::cout << "Queue is empty\n";
        } else {
            int i = front;  // Start from the front
            while (true) {
                std::cout << arr[i] << " ";  // Display each element
                if (i == rear) break;  // Stop when rear is reached
                i = (i + 1) % SIZE;  // Move circularly
            }
            std::cout << std::endl;
        }
    }
};
```

```
int main() {
    CircularQueue cq;  // Create a CircularQueue object
    cq.enqueue(10);  // Add element 10 to the queue
    cq.enqueue(20);  // Add element 20 to the queue
    cq.enqueue(30);  // Add element 30 to the queue
    cq.display();  // Display the queue
    cq.dequeue();  // Remove an element from the queue
    cq.display();  // Display the queue again
    return 0;
}
```