7. Implement Circular Queue using Circular Linked List. Perform following operations on it.
a) Insertion (Enqueue)
b) Deletion (Dequeue)
c) Display

```
#include <iostream>


class Node {
public:
    int data;
    Node* next;


    Node(int value) : data(value), next(nullptr) {}
};


class CircularQueue {
private:
    Node* front;
    Node* rear;


public:
    CircularQueue() : front(nullptr), rear(nullptr) {}


    bool isEmpty() {
        return front == nullptr;
    }


    void enqueue(int value) {
        Node* newNode = new Node(value);


        if (isEmpty()) {
            front = rear = newNode;
            rear->next = front;  // Circular linking
    } else {
        rear->next = newNode;
        rear = newNode;
        rear->next = front;  // Circular linking
    }

    std::cout << value << " enqueued to the queue." << std::endl;
}

void dequeue() {
    if (isEmpty()) {
        std::cout << "Queue is empty. Cannot dequeue." << std::endl;
        return;
    }

    int value = front->data;
    Node* temp = front;

    if (front == rear) {
        front = rear = nullptr;
    } else {
        front = front->next;
        rear->next = front;  // Circular linking
    }

    delete temp;
    std::cout << value << " dequeued from the queue." << std::endl;
}

void display() {
    if (isEmpty()) {
            std::cout << "Queue is empty." << std::endl;
            return;
        }

        Node* temp = front;
        do {
            std::cout << temp->data << " ";
            temp = temp->next;
        } while (temp != front);

        std::cout << std::endl;
    }
};

int main() {
    CircularQueue cq;

    cq.enqueue(10);
    cq.enqueue(20);
    cq.enqueue(30);
    cq.display();

    cq.dequeue();
    cq.display();

    cq.enqueue(40);
    cq.enqueue(50);
    cq.display();

    return 0;
}
```