Queue Implementation Using Linked List in C++

Objective:
The objective of this task is to understand and implement a Queue data structure using a Linked List in C++. This task demonstrates dynamic memory allocation and FIFO (First In First Out) behavior.

Problem Statement

Implement a Queue using a linked list with the following operations:
- Enqueue (insert an element at the rear)
- Dequeue (remove an element from the front)
- Display the elements of the queue

Concepts Used

- Queue Data Structure (FIFO)
- Linked List
- Dynamic Memory Allocation
- Pointers
- Classes and Objects

Source Code (C++)

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

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

        Node(int value) {
        data = value;
        next = NULL;
        }
```

```
};

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

public:
	Queue() {
	front = rear = NULL;
	}

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

	if (rear == NULL) {
	front = rear = newNode;
	return;
	}

	rear->next = newNode;
	  rear = newNode;
	}

	// Dequeue operation
	void dequeue() {
	if (front == NULL) {
	cout << "Queue is empty. Cannot dequeue." << endl;
	return;
	}

	Node* temp = front;
	front = front->next;

	if (front == NULL) {
	rear = NULL;
	}

	delete temp;
	}

	// Display queue
	void display() {
	if (front == NULL) {
	cout << "Queue is empty." << endl;
	return;
	}
```

```
        Node* temp = front;
        cout << "Queue elements: ";
        while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
        }
        cout << endl;
        }
};

int main() {
        Queue q;

        q.enqueue(10);
        q.enqueue(20);
        q.enqueue(30);

        q.display();

        q.dequeue();
        q.display();

        return 0;
}
```

Explanation of the Code

1. The Node class represents each element of the queue using a linked list node.
2. The Queue class maintains two pointers: front and rear.
3. enqueue() inserts a new element at the rear of the queue.
4. dequeue() removes an element from the front of the queue.
5. display() traverses the queue from front to rear and prints elements.
6. Dynamic memory allocation is done using the new keyword.
7. The main() function demonstrates enqueue and dequeue operations.

Program Output

```
student@student-virtual-machine:~/25SUB4508_LSP/25SUB4508_56133/Assignments/day19$ g++ queue_using_linked_list.cpp
student@student-virtual-machine:~/25SUB4508_LSP/25SUB4508_56133/Assignments/day19$ ./a.out
Queue elements: 10 20 30
Queue elements: 20 30
student@student-virtual-machine:~/25SUB4508_LSP/25SUB4508_56133/Assignments/day19$
```

Conclusion

This task demonstrates the implementation of a Queue using a linked list in C++. Using a linked list allows dynamic queue size without memory overflow issues. Queue data structures are widely used in scheduling, buffering, and real-time systems.