1. **Design, implement, and test a Queue data structure that:**
    1. uses a linked-list to store values in the queue

```
template <typename T>
class Node {
public:
    T data;
    Node<T>* next;

    Node(T val) : data(val), next(nullptr) {}
};
```

    2. has an enqueue method that will appropriately add a value to the back of the queue as an appropriate element

```
void enqueue(T val) {
    Node<T>* newNode = new Node<T>(val);
    if (rear == nullptr) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    size++;
}
```

3.has a dequeue method that will appropriately remove an element from the front of the queue and return its value

```
T dequeue() {
    if (isEmpty()) {
        throw std::out_of_range("Queue is empty");
    }
    T val = front->data;
    Node<T>* temp = front;
    front = front->next;
    if (front == nullptr) {
        rear = nullptr;
    }
    delete temp;
    size--;
    return val;
}
```

4.Optionally has a peek method that returns the value at the front of the queue without removing it

```
T peek() {
    if (isEmpty()) {
        throw std::out_of_range("Queue is empty");
    }
    return front->data;
}
```

Testing:

Enqueue:

```
75      // Test enqueue method
76      Queue<int> q;
77      q.enqueue(10);
78      q.enqueue(20);
79      q.enqueue(30);
80      std::cout << "Queue after enqueueing: ";
81      while (!q.isEmpty()) {
82          std::cout << q.dequeue() << " ";
83      }
84      std::cout << std::endl;
```

Dequeue:

```
86      // Test dequeue method
87      Queue<int> q2;
88      q2.enqueue(100);
89      q2.enqueue(200);
90      q2.enqueue(300);
91      std::cout << "Dequeueing from queue: ";
92      std::cout << q2.dequeue() << std::endl;
93
```

Peek:

```
94      // Test peek method
95      Queue<int> q3;
96      q3.enqueue(1000);
97      q3.enqueue(2000);
98      q3.enqueue(3000);
99      std::cout << "Peeking at front element: ";
100     std::cout << q3.peek() << std::endl;
101
```