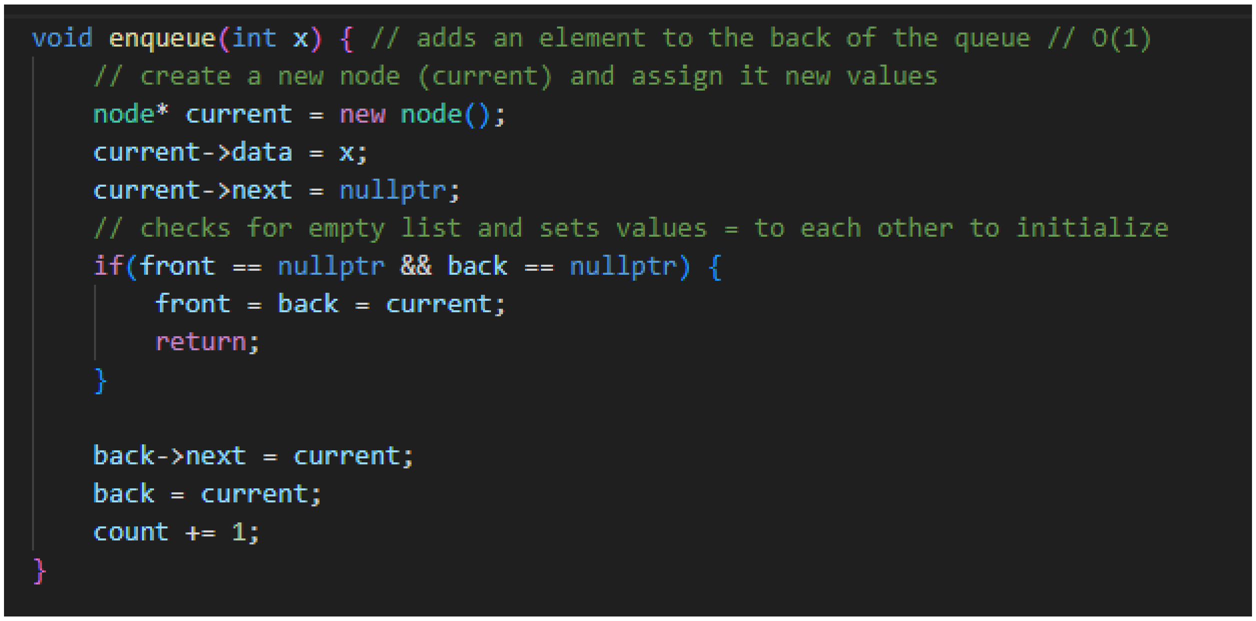
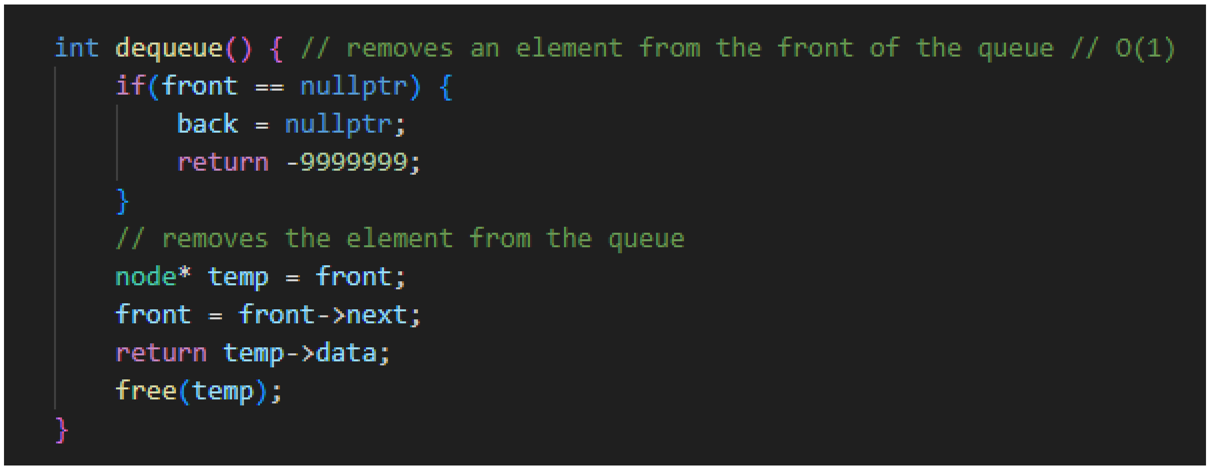
1. Based on what we know about linked lists, stacks, and queues, design a linked queue (a queue using a linked-list to store the data in the structure)
2. Design, implement, and test a Queue data structure that:
    1. uses a **linked-list** to store values in the queue
    2. has an **enqueue** method that will appropriately **add** a value to the **back** of the queue as an appropriate element



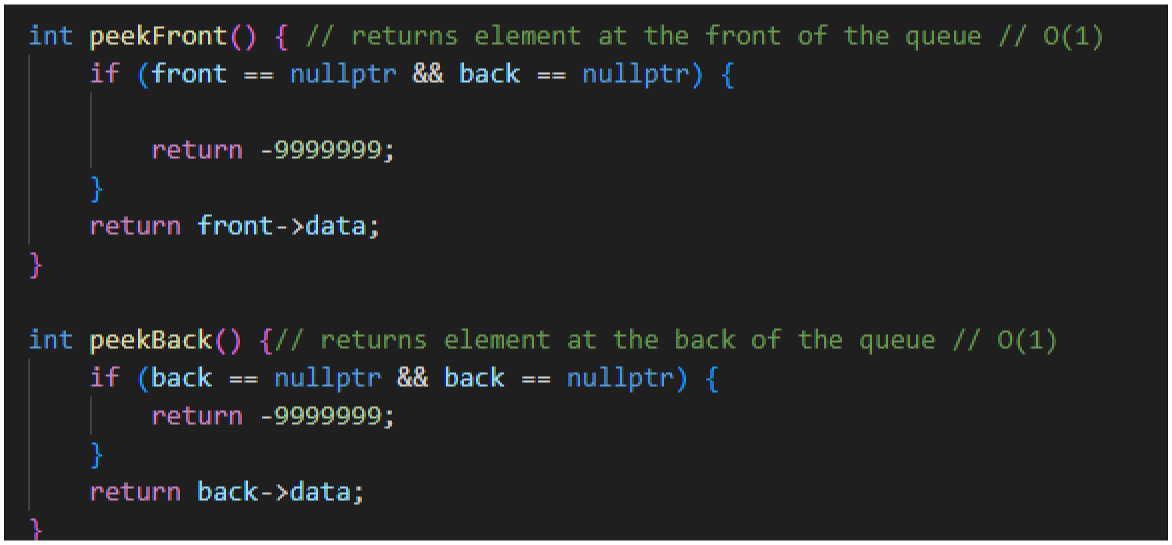
```
void enqueue(int x) { // adds an element to the back of the queue // O(1)
    // create a new node (current) and assign it new values
    node* current = new node();
    current->data = x;
    current->next = nullptr;
    // checks for empty list and sets values = to each other to initialize
    if(front == nullptr && back == nullptr) {
        front = back = current;
        return;
    }

    back->next = current;
    back = current;
    count += 1;
}
```

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



```
int dequeue() { // removes an element from the front of the queue // O(1)
    if(front == nullptr) {
        back = nullptr;
        return -9999999;
    }
    // removes the element from the queue
    node* temp = front;
    front = front->next;
    return temp->data;
    free(temp);
}
```

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



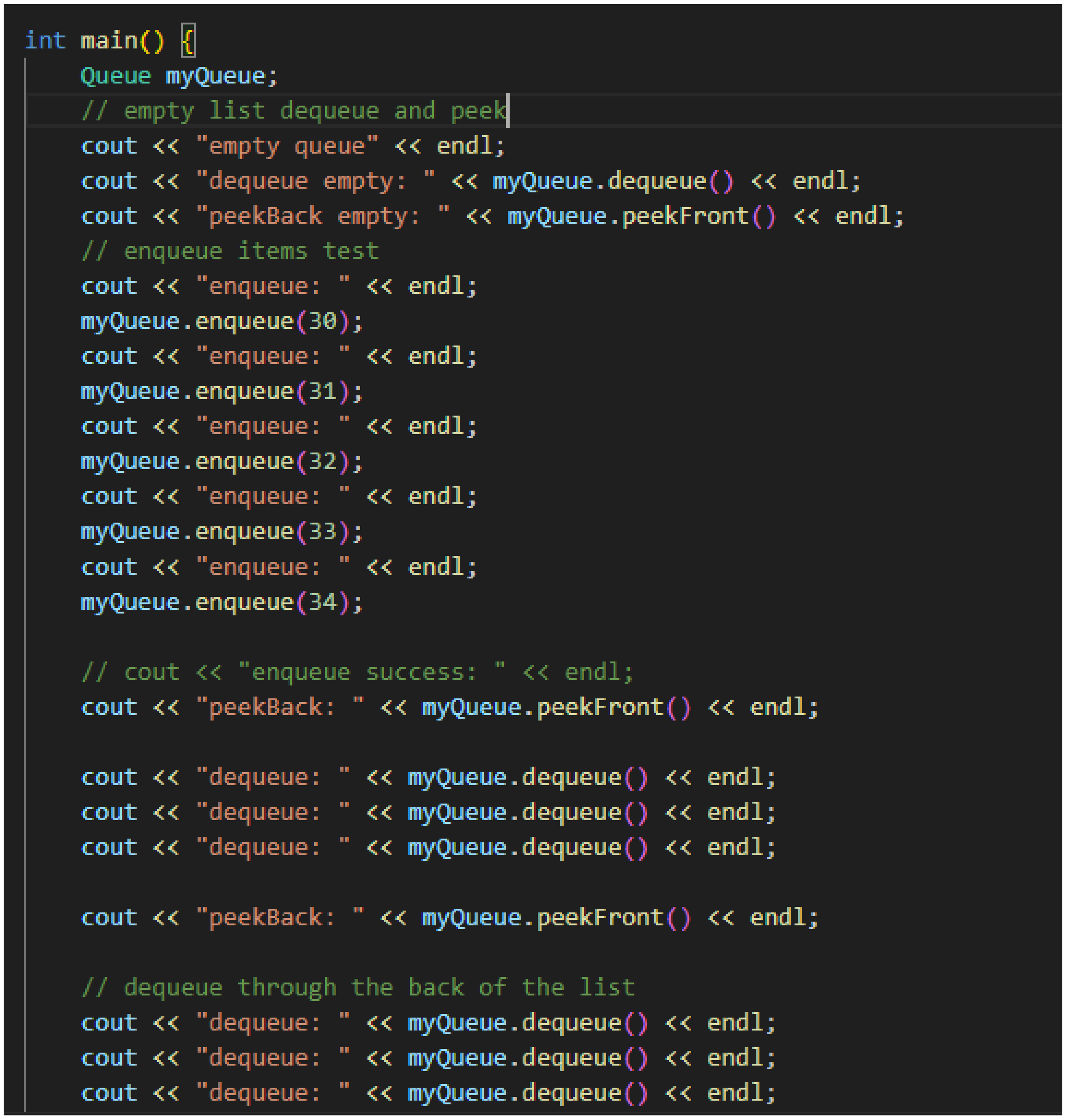
```
int peekFront() { // returns element at the front of the queue // O(1)
    if (front == nullptr && back == nullptr) {

        return -9999999;
    }
    return front->data;
}

int peekBack() {// returns element at the back of the queue // O(1)
    if (back == nullptr && back == nullptr) {
        return -9999999;
    }
    return back->data;
}
```

Bonus if you also create an array based Queue!

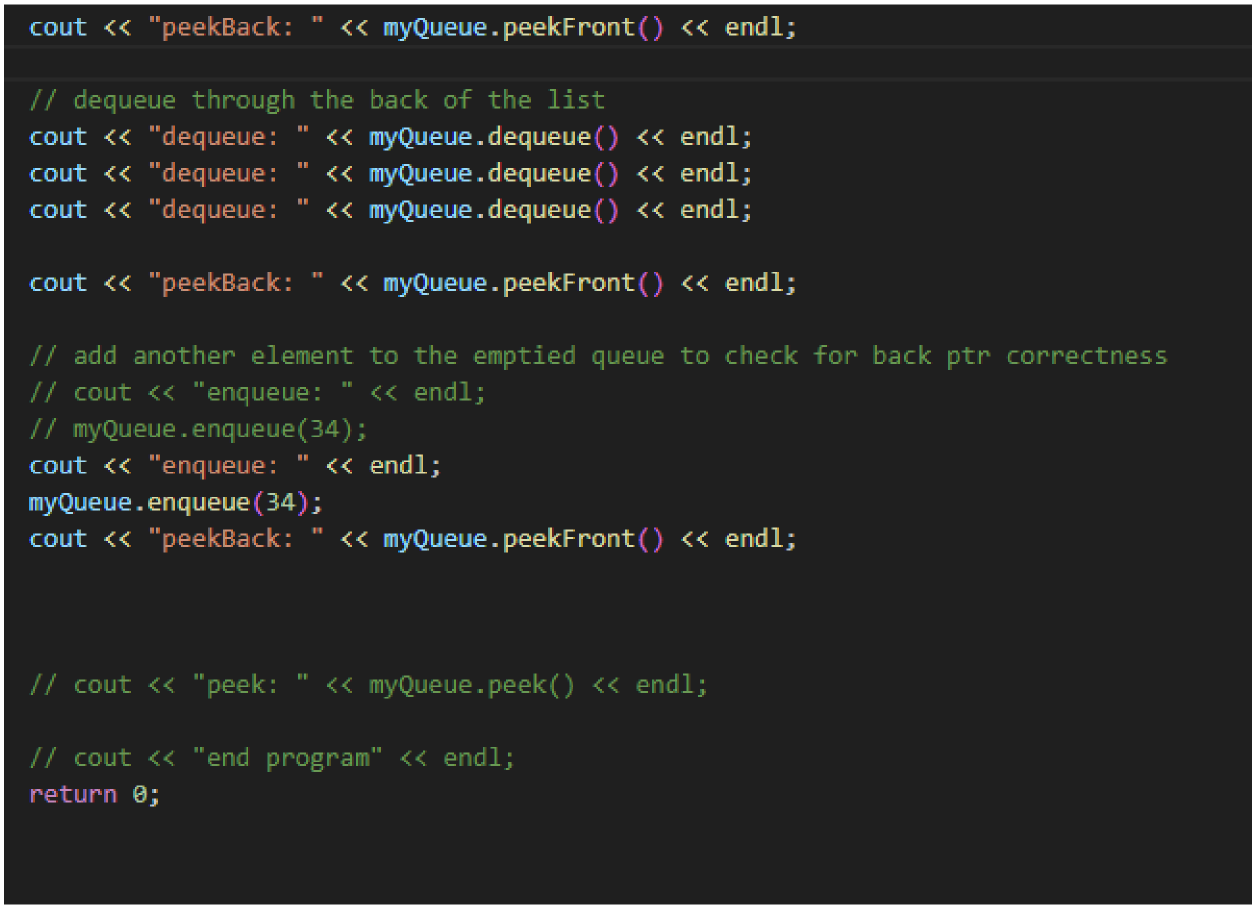
3. Analyze the complexity of your implementations (at least the run-time of the add, remove, and peek methods).
   (**Note** that we will often consider operations not having to do with the structure as O(1), even if they might be expensive operations in terms of real-time or space used)
   (**Note** that if you are not in class when we talk about Asymptotic Big-O notation, you can find tons of good examples online)

All the functions I implemented were O(1) because instead of iterating throught the linked list to get to the values, I made a front and back pointer that always updated to the proper value after any given operation, making it take up slightly more storage but greatly increasing the runtime efficiency which would have been O(N) if I iterated through.

4. Tests: Be sure to include at least one test for each piece of functionality that should verify that your code is working!



```cpp
int main() {
    Queue myQueue;
    // empty list dequeue and peek
    cout << "empty queue" << endl;
    cout << "dequeue empty: " << myQueue.dequeue() << endl;
    cout << "peekBack empty: " << myQueue.peekFront() << endl;
    // enqueue items test
    cout << "enqueue: " << endl;
    myQueue.enqueue(30);
    cout << "enqueue: " << endl;
    myQueue.enqueue(31);
    cout << "enqueue: " << endl;
    myQueue.enqueue(32);
    cout << "enqueue: " << endl;
    myQueue.enqueue(33);
    cout << "enqueue: " << endl;
    myQueue.enqueue(34);

    // cout << "enqueue success: " << endl;
    cout << "peekBack: " << myQueue.peekFront() << endl;

    cout << "dequeue: " << myQueue.dequeue() << endl;
    cout << "dequeue: " << myQueue.dequeue() << endl;
    cout << "dequeue: " << myQueue.dequeue() << endl;

    cout << "peekBack: " << myQueue.peekFront() << endl;

    // dequeue through the back of the list
    cout << "dequeue: " << myQueue.dequeue() << endl;
    cout << "dequeue: " << myQueue.dequeue() << endl;
    cout << "dequeue: " << myQueue.dequeue() << endl;
```



```
cout << "peekBack: " << myQueue.peekFront() << endl;

// dequeue through the back of the list
cout << "dequeue: " << myQueue.dequeue() << endl;
cout << "dequeue: " << myQueue.dequeue() << endl;
cout << "dequeue: " << myQueue.dequeue() << endl;

cout << "peekBack: " << myQueue.peekFront() << endl;

// add another element to the emptied queue to check for back ptr correctness
// cout << "enqueue: " << endl;
// myQueue.enqueue(34);
cout << "enqueue: " << endl;
myQueue.enqueue(34);
cout << "peekBack: " << myQueue.peekFront() << endl;



// cout << "peek: " << myQueue.peek() << endl;

// cout << "end program" << endl;
return 0;
```

5. Be sure to commit changes regularly to your git repo
6. Once you have implemented and tested your code, add to the README file what line(s) of code or inputs and outputs show your work meeting each of the above requirements (or better, include a small screen snip of where it meets the requirement!).

7. Remember to submit a link to this project in Moodle to remind us to grade it!