Quentin Creese
Jess
CS 260
Assignment 03
Due: 01/29/24

(Linked Queue)

1. Uses a linked-list to store values in the queue:

```
// LinkedQueue constructor
LinkedQueue::LinkedQueue() : front(nullptr), rear(nullptr) {}
```

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

```
// Enqueue operation
void LinkedQueue::enqueue(int value) {
   Node* newNode = new Node(value); // Create a new node with the provided
value
   if (!front) { // If the queue is empty
       front = newNode; // Set both front and rear pointers to the new node
       rear = newNode;
   } else { // If the queue is not empty
       rear->next = newNode; // Set the next pointer of the current rear node
to the new node
       rear = newNode; // Update the rear pointer to the new node
   }
}
```

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

```
// Dequeue operation
int LinkedQueue::dequeue() {
   if (!front) // If the queue is empty
       throw out_of_range("Queue is empty"); // Throw an exception
   int value = front->value; // Get the value of the front node
   Node* temp = front; // Store the current front node
   front = front->next; // Move front pointer to the next node
   delete temp; // Delete the previous front node
   if (!front) // If the queue becomes empty after dequeue
       rear = nullptr; // Update the rear pointer to nullptr
   return value; // Return the dequeued value
```

4. Optional has a peek method that returns the value at the front of the queue w/o removing it

```
// Peek operation
int LinkedQueue::peek() {
```

```
    if (!front) // If the queue is empty
        throw out_of_range("Queue is empty"); // Throw an exception
    return front->value; // Return the value of the front node
}
```

Bonus: Create an array-based Queue!

```
// Constructor initializes the queue with default capacity and sets front and rear to
-1
ArrayQueue::ArrayQueue() : capacity(1), queue(new int[capacity]), front(-1), rear(-1)
{}

// Destructor deallocates memory used by the queue
ArrayQueue::~ArrayQueue() {
    delete[] queue;
```

Testing:

```

#include <iostream>
#include <cassert>
#include "LinkedQueue.hpp"
#include "ArrayQueue.hpp"

using namespace std;

// Test cases for LinkedQueue class
void testLinkedListQueue() {
    LinkedQueue linkedQueue;

    // Test enqueue and peek operations
    linkedQueue.enqueue(1);
    assert(linkedQueue.peek() == 1); // Check if the front element is 1

    linkedQueue.enqueue(2);
    linkedQueue.enqueue(3);
    assert(linkedQueue.peek() == 1); // Check if the front element is still 1

    // Test dequeue operation
    assert(linkedQueue.dequeue() == 1); // Check if the dequeued element is 1
    assert(linkedQueue.dequeue() == 2); // Check if the dequeued element is 2
    assert(linkedQueue.dequeue() == 3); // Check if the dequeued element is 3
```

```
    // Test dequeue operation on an empty queue
    try {
        linkedQueue.dequeue();
    } catch (const out_of_range& e) {
        // Check if the expected exception is thrown when dequeueing from an empty
queue
        assert(string(e.what()) == "Queue is empty");
    }
}

// Test cases for ArrayQueue class
void testArrayQueue() {
    ArrayQueue arrayQueue;

    // Test enqueue and peek operations
    arrayQueue.enqueue(1);
    assert(arrayQueue.peek() == 1); // Check if the front element is 1

    arrayQueue.enqueue(2);
    arrayQueue.enqueue(3);
    assert(arrayQueue.peek() == 1); // Check if the front element is still 1

    // Test dequeue operation
    assert(arrayQueue.dequeue() == 1); // Check if the dequeued element is 1
    assert(arrayQueue.dequeue() == 2); // Check if the dequeued element is 2
    assert(arrayQueue.dequeue() == 3); // Check if the dequeued element is 3

    // Test dequeue operation on an empty queue
    try {
        arrayQueue.dequeue();
    } catch (const out_of_range& e) {
        // Check if the expected exception is thrown when dequeueing from an empty
queue
        assert(string(e.what()) == "Queue is empty");
    }
}

int main() {
    // Run tests for LinkedQueue
    cout << "Running tests for LinkedQueue..." << endl;
    testLinkedListQueue();
    cout << "All tests for LinkedQueue passed!" << endl;
```

```
    // Run tests for ArrayQueue
    cout << "Running tests for ArrayQueue..." << endl;
    testArrayQueue();
    cout << "All tests for ArrayQueue passed!" << endl;

    return 0;
}
```

```
Running tests for LinkedQueue...
All tests for LinkedQueue passed!
Running tests for ArrayQueue...
All tests for ArrayQueue passed!
```

Running main.cpp:

Enqueueing:

```
Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Exit
Enter your choice: 1
Enter value to enqueue: 1
Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Exit
Enter your choice: 1
Enter value to enqueue: 10
Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Exit
Enter your choice: 1
Enter value to enqueue: 100
```

Peeking:

```
Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Exit
Enter your choice: 3
Front element of LinkedQueue: 1
Front element of ArrayQueue: 1
```

Dequeueing:

```
Enter your choice: 2
Dequeued value from LinkedQueue: 1
Dequeued value from ArrayQueue: 1
Menu:
1. Enqueue
2. Dequeue
3. Peek
4. Exit
Enter your choice: 2
Dequeued value from LinkedQueue: 10
Dequeued value from ArrayQueue: 10
```

Peeking:

```
Enter your choice: 3
Front element of LinkedQueue: 100
Front element of ArrayQueue: 100
```

Dequeueing empty node:

```
Enter your choice: 2
Dequeued value from LinkedQueue: Queue is empty
```

Exit:

```
Enter your choice: 4
Exiting program.
```