UNIVERSITY LOGO DEPT LOGO

SEMESTER INFORMATION [CLASS NAME]

**Linked List Project**

# Overview

This linked list project is intended to provide students with an understanding of how sequential list data structures function. This project involves the implementation of a templated, doubly-linked list with methods that allow for its use as a queue or stack container.

# Structure

A doubly linked list is a type of linked list which is linked in both directions, pointing to the next and previous nodes in the list. It usually terminates, at both ends, in pointers to nullptr. Depending on how one adds to or remove items from either end, the linked list can behave either as a stack or as a queue.

A linked list is made up of ***nodes***. Each node in the list contains some data (in this case, a location represented by a pair of coordinates) and a pointer to the next and previous nodes in the list. The first node in the list is called the *front*, and the last node is called the *back*.

**A linked list with location coordinates as data**

5, 12

7, 24

8, 15

3, 4

*nullptr*

*nullptr*

*back*

*front*

Write the LinkedList and LinkedList::Iterator classes in the *LinkedList.h* file with these methods:

LinkedList<T>::Iterator

*Iterator()*

This is the constructor for **Iterator**.

*T operator\*() const*

Return the element at the iterator's current position in the queue.

*Iterator& operator++()*

Pre-increment overload; advance the operator one position in the list. Return this iterator. ***NOTE***: if the iterator has reached the end of the list (past the last element), its data should be equal to **LinkedList<T>::end()**.

*bool operator==(Iterator const& rhs)*

Returns true it both iterators point to the same node in the list, and false otherwise.

*bool operator!=(Iterator const& rhs)*

Returns false it both iterators point to the same node in the list, and true otherwise.

LinkedList<T>

*LinkedList<T>()*

This is the constructor for **LinkedList**.

*Iterator begin() const*

Return an Iterator pointing to the beginning of the list.

*Iterator end() const*

Return an Iterator pointing past the end of the list (an invalid, unique state, data likely pointing to **nullptr**.)

*bool isEmpty() const*

Return true if there are no elements, false otherwise.

*T getFront() const*

Return the first element in the list.

*T getBack() const*

Return the last element in the list.

*void enqueue(T element)*

Insert the specified element to the list. Handle the special case when the list is empty.

*void dequeue()*

Remove the first element from the list. Handle the special case when it is also the last element.

*void pop()*

Remove the last element from the list. Handle the special case when it is also the first element.

*void clear()*

Remove all elements from the list.

*bool contains(T element) const*

Return true if you find a node whose data equals the specified element, false otherwise.

*void remove(T element)*

Remove the first node you find whose data equals the specified element. Be sure to update the pointers appropriately. Test your code for the following scenarios:

* If you remove the first node from the list
* If you remove a node from the middle of the list
* If you remove the last node from the list
* If you remove the only node from the list

# Submissions

Submit the following files at the end of this project:

* LinkedList.h

Place them in the ROOT directory of your zip file, not in a subdirectory. Do not submit any other files.