UF UNIVERSITY of FLORIDA
Herbert Wertheim
College of Engineering

COMPUTING FOR LIFE
COP3504C

Lab 09: Linked List

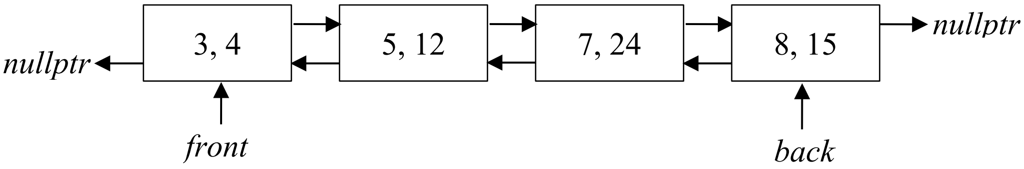
Overview

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.

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



Specification

Students have been provided with a test driver program (**`main.cpp`**), build file (**`CMakeLists.txt`**), and built-in memory leak detection (**`nvwa`**). Full credit requires implementation with <u>no errors or warnings</u>.

Classes

Students will write a linked list class and a nested iterator class for the linked list as follows.

<u>`LinkedList<T>::Iterator`</u>

`public T operator*() const`
Return the element at the iterator's current position in the queue.

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

`Iterator& operator--()`
Pre-decrement overload; recedes one element. Return this iterator. ***<u>NOTE</u>***: if the iterator has reached the end of the list (before the first element), its data should be equal to **`LinkedList<T>::end()`**.

`bool operator==(Iterator const& rhs)`
Return **`true`** it both iterators point to the same node in the list, and **`false`** otherwise.

`bool operator!=(Iterator const& rhs)`
Return **`false`** it both iterators point to the same node in the list, and **`true`** otherwise.

LinkedList<T>

public **LinkedList**<T>()
Construct a new **LinkedList**<T>.

Iterator **begin**() **const**
Return an **Iterator** pointing to the beginning of the list.

Iterator **tail**() **const**
Return an **Iterator** pointing to the last node 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.

bool **contains**(T element) **const**
Return **true** if list contains a node whose data equals the specified element and **false** otherwise.

void **enqueue**(T element)
Adds the specified element to the back of the list.

void **dequeue**()
Remove the first element from the list.

void **pop**()
Remove the last element from the list.

void **clear**()
Removes all elements from the list.

void **remove**(T element)
Remove the first node found whose data equals the specified element. ***Note:*** be sure to update the pointers appropriately; test your code for the following scenarios:

- Remove the first node from the list
- Remove a node from the middle of the list
- Remove the last node from the list
- Remove the only node from the list

Submissions

NOTE: Your output must match the example output *exactly*. If it does not, ***you will not receive full credit for your submission***!

Files: LinkedList.h
Method: Submit on ZyLabs