The design of my doubly linked list is simple. There is no dummy node and it is not circular. There is a head pointer pointed at the very first object and a tail pointer pointed at the last object. The Node pointer prev on the first item is nullptr, signifying nothing comes before it in the list. The Node pointer next on the last item is nullptr, showing that nothing comes after it in the list.

Psuedocode:

Map::erase

Find the word we are looking for with a loop

If the item is not in the list

Return false

If the item is the only node in the list

Delete the item

Else If the item is the first node in the list

Delete the item

Else If the item is the last node in the list

Delete the item

Else the item is somewhere in the middle of the list

Delete the item

Reduce the size counter member variable

Return true

combine

Erase all items from the result list

Add items from m1 to result

Add items from m2 to result

If an item from m2 is already in result

If the Value in those items is not the same

Remove the item from result

If we had to remove an item from result because It was already in the list, return false

Otherwise return true;

subtract

Remove all items from result

Add items from m1 into result

Compare items from m2 and result.

If items share the same key, remove the item with that key from result

#include "Map.h"

#include <cassert>

using namespace std;

int main() {

Map m;

assert(m.empty()); //checking empty function

assert(m.insert("A", 1)); //checking insert function

assert(!m.empty());

assert(!m.insert("A", 1)); //checking insert with a duplicate key

assert(m.insert("B", 2));

assert(m.insert("C", 3));

assert(m.size() == 3); //checking size

assert(m.contains("A")); //checking contains

assert(!m.contains("D"));

assert(m.update("B", 4)); //checking update

assert(m.insertOrUpdate("D", 4));//checking insert or update

assert(m.insertOrUpdate("D", 5));

assert(!m.erase("E")); //checking erase

assert(m.erase("A"));

assert(m.erase("C"));

assert(m.erase("D"));

assert(m.insert("E", 5));// checking insert works after erase

assert(m.insert("F", 6));

double v = 0;

string k = "Z";

assert(!m.get("A", v) && v == 0); //checking the first get

assert(m.get("F", v) && v == 6);

assert(m.size() == 3);

assert(!m.get(3, k, v) && v==6 && k=="Z"); //checking the second get

assert(m.get(0, k, v) && v == 4 && k == "B");

assert(m.get(1, k, v) && v == 5 && k == "E");

assert(m.get(2, k, v) && v == 6 && k == "F");

Map m2(m); //copy constructor works

assert(m2.insert("G", 7)); //making sure that the new map is new and doesnt just point to the old one

m = m2; //equal operator works

assert(m2.erase("B")); //making sure the new map is new and doesnt just point to the old one

Map m3;

assert(m3.insert("X", 10));

assert(m3.insert("Y", 15));

assert(m3.insert("H", 17));

assert(m3.insert("Z", 20));

assert(combine(m, m2, m3)); //checking combine

assert(m2.update("G", 9));

assert(!combine(m, m2, m3)); //checking combine

assert(m2.erase("E"));

assert(m2.insert("X", 9));

subtract(m, m2, m3); //checking subtract

m.swap(m2); //checking swap

}