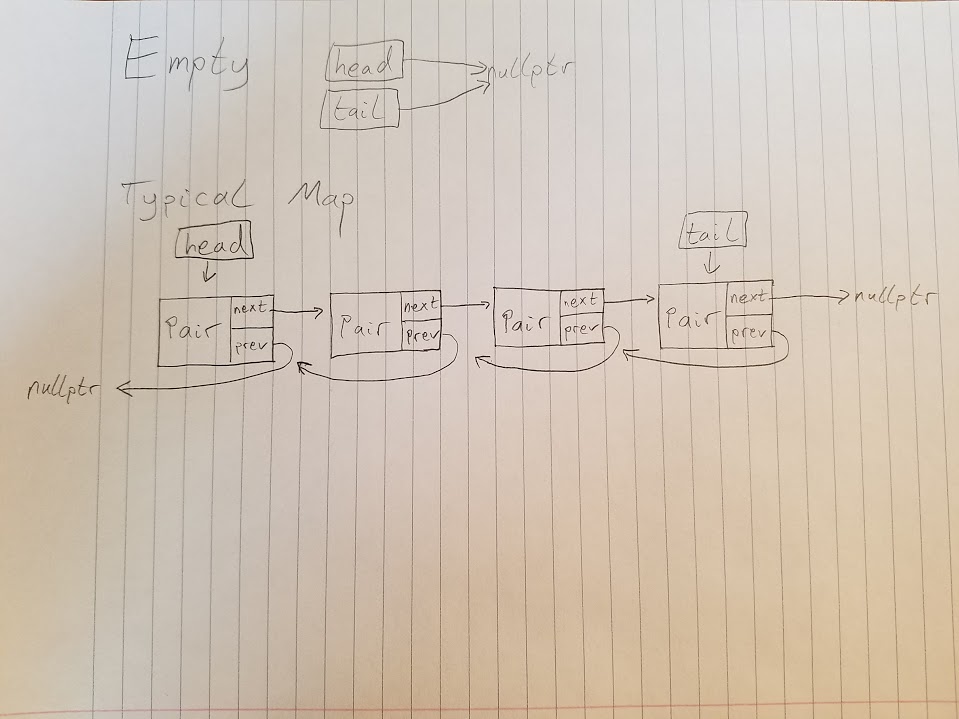
Report Project 2

1. Design

My doubly linked list design is simple in that it is not circular nor has dummy node. An empty list, as shown in the picture, points both head and tail to nullptr. Inserting a new node adds it to the end. All traversals are done from head to tail. Each node contains the next and previous pointers as well as a pair with a keyType and valueType variable each.



1. Pseudocode

**Insert**

…

Return false if the map already contains key

Special case for an empty Map

Allocate a Node, set its values, and set head ptr to the Node

Otherwise,

Go to tail

allocate a new node referenced by the last next ptr

set its values

reset tail ptr to the new Node

increment size if something is inserted

…

**Update**

…

Run through entire map until the key is found

Update that key’s value

If never found, return false

…

**Erase**

…

If empty or nothing matches, false

If only one node, delete it, reformat list, decrement size, true

Until the end,

Find a match

Delete the match, reformat list

Just in case you get through without finding a match, return false again

…

**Swap**

…

Swap both Maps’ sizes

Swap both Maps’ head pointers

Swap both Maps’ tail pointers

…

**Combine**

…

Set result to the first parameter

Loop until end of second parameter

Compare each value of second parameter to every node in the result

If any repeats, delete from result and return false

Otherwise, insert the node from second parameter into result

Return true if no repeats

…

**Subtract**

…

Set the result to first parameter

Loop through the result

If anything in second parameter matches,

Erase from result

…

1. Test Cases

From strings to doubles

#include "Map.h"

#include <iostream>

#include <cassert>

using namespace std;

void test()

{

Map m; //test constructor

assert(m.size() == 0); //test empty size

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

assert(m.insert("August", 123)); //test insert

assert(m.insert("Amanda", 456)); //insert another

assert(!m.insert("August", 234)); //can't insert a key already in map

assert(!m.insert("August", 123)); //can't insert a key AND value already in map

assert(m.size() == 2); //check random size

ValueType v = 42;

assert(!m.get("John", v) && v == 42); //check 2 par. get on key not in list and doesn't modify par.

assert(m.get("August", v) && v == 123); //check 2 par. get on valid key and does modify par.

v = 42;

KeyType x = "Lucy";

assert(m.get(0, x, v) && ((x == "August" && v == 123) || (x == "Amanda" && v == 456))); //check 3 par. get on first element and does modify par.

KeyType x2 = "Ricky";

assert(m.get(1, x2, v) && ((x2 == "August" && v == 123) || (x2 == "Amanda" && v == 456)) && x != x2); //check 3 par. get on last element and does modify par.

assert(m.insert("Steve", 123)); //insert a new key with a value already in map

assert(!m.update("John", 72.5)); //can't update a key not already in map

ValueType val;

assert(m.update("August", 96.45) && m.get("August", val) && val == 96.45); //update a value and use get to make sure it updated

assert(m.insertOrUpdate("John", 73)); //inserts a key when not already in map

KeyType s;

assert(m.get(3, s, val) && (s == "John") && (val == 73)); //make sure John was inserted properly using 3 par get function

assert(m.size() == 4); //check size function again

assert(m.insertOrUpdate("August", 91) && m.get(0, s, val) && s == "August" && val == 91); //insertorupdate updates properly if key already exists

assert(m.contains("Amanda")); //test valid contains

assert(!m.contains("Joe")); //test invalid contains

assert(m.erase("Steve") && m.size() == 3); //test valid erase call

assert(!m.erase("Timothy") && m.size() == 3); //test invalid erase call

Map m2; //create a new map to test swap and subtract

assert(m2.insert("Steve", 40));

assert(m2.insert("Liz", 21));

m.swap(m2);

assert(!m.contains("August")); //make sure August is no longer in m

assert(m.contains("Steve")); //check if a value from m2 is now in m

assert(m2.contains("August")); //check if a value from m is now in m2

assert(m.insert("August", 123)); //test insert

assert(m.insert("Amanda", 456)); //insert another

assert(m.size() == 4); //sizes are swapped

Map m3 = m2; //copy constructor

Map combineResult;

Map subtractResult;

subtract(m, m2, subtractResult); //subtract m2 from m

assert(subtractResult.size() == 2); //size should be 2. It deletes "August" and "Amanda"

combine(m, m2, combineResult);

assert(combineResult.size() == 4);

}

int main()

{

test();

cout << "Passed all tests" << endl;

}