Elon Glouberman

ID: 904769785

**Computer Science Project 2**

**Description of Design**

For my Map class, I implemented a private struct called “Node” that contained a keyType key, a valueType value, a Node pointer called next, and a node pointer called prev. Outside the struct, as private variables, I had a head node, that was a node pointer and I also had int m\_size that held the size of my map. The head node is automatically initialized to nullptr and as you insert elements into the list, the last node always points to the nullptr. The order of insert is when you add something to the list, it appends that element to the end of the list.

**Pseudocode for non-trivial algorithms**

Void Combine function (const map m1, const map m2, result)

Set bool shouldReturn to true

Set result to an empty map

For loop that iterates through m1

Have keyType k1 and ValueType v1

Use “get” function to set k1 and v1 to index of m1

For loop that iterates through m2

Have keyType k2 and ValueType v2

Use “get” function to set k2 and v2 to index of m2

If both keys are equal but values aren’t, return false through bool variable “shouldReturn”

If both pairs are exactly the same, we insert both into result

If both pairs are not at all the same

Check if either of the pairs has an equal counterpart in the other map

Continue;

We insert both into result

We return “shouldReturn”

Void Subtract function (const map m1, const map m2, result)

Set result to empty map

Iterate through all of m1 and see if m2 has index i

Have keyType k1 and valueType v1 and we check if m2 has k1 through the use of the get function

If m2 doesn’t contain k1, a key in m1

Insert k1 into result

Return;

Bool Map:: insert(const keyType & key, const valueType & value)

If head is the only item in the list (if it is == nullptr)

Make new Node\*, p and we set its value, key to the parameters

We set p->next to nullptr

We set p>prev to head

Make m\_size = 1

Return true

Node\* p = head. If p is not nullptr (if p is not at the end of the list)

Check if p’s key is == to key in the parameter to see if key already exists in the map

If it does, return false

Node\* j = head, where we check while j’s next value != nullptr

J = j->next to get j to the last index

We insert new node at the last index, update key, value, new node’s next pointing to nullptr, its previous at j and we increment the size

Bool Map:: erase(const keyType& key)

If key is in the map

Assign Node\* p as head

If the first element in the map is the key we want to erase

If that key is the only element in the map

Empty the map. Head assigned to nullptr

P’s previous is nullptr

Deallocate p

Decrease size

P = nullptr

Return true

Set the element after p’s prev to nullptr

Set head to the next element

Deallocate p

Decrease size

Element we want to erase is not first, so we have while p is not nullptr

If p is not nullptr and the next value after p is the one we want to erase

Break

P moves on to next element

If we found the element we want to erase

Assign temp variable killme to the key we want to erase

Restring p’s next to the element after killme

Restring the element after killme’s prev to p

Deconnect the pointers in killme

Deallocate killme

Size decreases

Return true

If key not in map return false

//

//  main.cpp

//  CS32\_proj2

//

//  Created by Elon Glouberman on 1/25/18.

//  Copyright (c) 2018 Elon Glouberman. All rights reserved.

//

#include <iostream>

#include "Map.h"

#include <cassert>

int main() {

    // default constructor

    Map m;

    // For an empty map:

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

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

    assert(!m.erase("Ricky"));  // nothing to erase

    //default constructor

    Map k;

    assert(k.insert("Fred", 123.3)); //we insert Fred

    assert(k.erase("Fred"));//erases a map with only one item

   assert(k.insert("Fred", 123.3));

    assert(k.insert("Ethel", 456.1)); // we insert Ethel

    assert(k.size()==2); // make sure size is == 2

    assert(!k.insert("Fred", 456.1)); //Fred already exists, so this should return false

    assert(k.erase("Ethel"));  // we erase Ethel

    assert(k.size()==1); // only one element in our map

    // we insert two pairs

    k.insert("Lucy", 789);

    k.insert("george", 34.2);

    //erase first element

    assert(k.erase("Fred")); //erases "Fred", element 1

    assert(k.insert("Fred", 123.3)); // we put it back

    //erase middle element

    assert(k.erase("Lucy")); //erases "Lucy", element 2

    assert(k.insert("Lucy", 789)); // we put it back

    //erase last element

    assert(k.erase("george")); //erases "george", element 3 (last element)

    assert(k.insert("george", 34.2)); //we put it back

    //check contains function

    assert(k.contains("Lucy") && !k.contains("Ethel"));// make sure Lucy is in the map and Ethel doesn't exist

    //check update function

    assert(k.update("george", 45.7));// we change george's value from 34.2 to 45.7

    double k2; // we initialize k2 as a double

    //check if updated using "get" function

    assert(k.get("george",k2) && k2 == 45.7); // we check if george's value is changed to 45.7

    //insertOrUpdate always returns true

    assert(k.insertOrUpdate("", 45));

    assert(k.contains(""));

    assert(k.size() ==4); //size shoud be 4

    assert(k.insertOrUpdate("george", 3)); //should call the update function

    assert(k.size() ==4); //size shoud still be 4

    //Map m has size of 0, with no elements

    //Map k has a size of 4

    //testing swap function

    k.swap(m);

    //k should have size of 0 now

    assert(k.size() ==0);

    assert(m.size() == 4 && m.contains("george") && m.contains("Lucy") && m.contains("Fred") && m.contains(""));

    //testing = operator

    Map equals;

    equals = m; // we assign Map equals to m

    assert(equals.size() == 4 && equals.contains("george") && equals.contains("Lucy") && equals.contains("Fred") && equals.contains(""));

    //testing combine and subtract functions

    //default constructor for map c

    Map c; // initialize c with Lucy, Fred, and Ethel

    c.insert("Lucy", 789);

    c.insert("Fred", 123);

    c.insert("Ethel", 456);

    Map j; // initializing j with Lucy and Ricky

    j.insert("Lucy", 789);

    j.insert("Ricky", 321);

    assert(combine(j,c,m)); // combining should get us Fred 123, Lucy 789, Ricky 321, Ethel 456

    assert(m.size() == 4);

    assert(m.contains("Fred") && m.contains("Lucy") && m.contains("Ricky") && m.contains("Ethel"));

    j.update("Lucy",654);  // we update Lucy to have a different value.

    //m is not empty here when we pass it through combine

    assert(!combine(j,c,m) && !m.contains("Lucy")); // should return false, and Lucy should be excluded

    //testing the subtract function

    assert(j.update("Lucy",789));

    assert(j.insert("Ethel", 654));

    subtract(c,j,m); // this should return m with only one pair of Fred 123

    assert(m.size()==1 && m.contains("Fred")); // size is 1, with Fred being the only element in it

    std::cout<< "Passed all tests!" << std::endl;

    return 0;

}