Description of Doubly Linked List

For the doubly linked list created for my Map, no dummy nodes were used. However, after all the implementations, I realized that creating a dummy header node before the head of the list is a good practice since this can ensure that the next node exists for every element.

My doubly linked list is not circular. My list nodes contain information about their own data, including keys and values, and they also contain pointers to the previous and next nodes. They are not in any particular order.

Examples of a typical Map and an empty Map are illustrated below:

Typical Map:

NULL

Next

Previous

­

Next

Previous

Next

Previous

Next

Previous

NULL

Key / Value

Key / Value

Key / Value

Key / Value

Key / Value

Empty Map:

tail:

NULL

head:

NULL

Size:

0

Pseudocode for non-trivial functions:

//Copy Constructor

Map::Map(const Map& other)

{

Set size as the size of the other map

If the other map is empty,

Set the new linked list as empty and return

Create a new linked list

Repeatedly

Loop through the other map

Copy key & value each node of the other map and store into newNode

Arrange pointers and the tail node

}

//destructor

Map::~Map()

{

If the map is non-empty

Create a node ptr to traverse through the list

Starting from head

Repeatedly

Point ptr to the next node

Delete the previous node of ptr

Delete ptr

Set head and tail to nullptr

Set size to 0

}

//Insertion

bool Map::insert(const KeyType& key, const ValueType& value)

{

If this is an empty list

Create a new head

Set tail as the same as head

Store key/value to head

If not

Create a node ptr to traverse through the list

Repeatedly

Point ptr to the next node

Check if the key is already in the list

If the key is new

Assign key/value to the next node of ptr

Increase size by 1

}

//update

bool Map::update(const KeyType& key, const ValueType& value)

{

If the list is not empty

Create a node ptr to traverse through the list

Starting from head

Repeatedly

Point ptr to the next node

Check if the key matches any key in the list

If found

Update the old value of ptr with the given value

}

//erase

bool Map::erase(const KeyType& key)

{

If the list is not empty

Create a node ptr initially at head

If want to erase the first node / head

If the list has more than one node

Delete ptr and rearrange the whole list

If the list has only one node

Delete ptr and empty the whole list

Else

Repeatedly

Find the position where the signal key is presented

Delete the ptr

//contains

bool Map::contains(const KeyType& key) const

{

If the list is not empty

From head to tail

Check if the key is presented in the list

Return true if yes, and vice versa

}

//first get

bool Map::get(const KeyType& key, ValueType& value) const

{

If the list is not empty

From head to tail

Traverse through the list to find the key

If found, copy and assign the corresponding value

}

//second get

bool Map::get(int i, KeyType& key, ValueType& value) const

{

If i is a valid integer and the list is not empty

Create two new nodes

Repeatedly

Use bubble sort algorithm to sort the list alphabetically

Find the ith element in the sorted linked list

Copy and store key and value

}

//swap

void Map::swap(Map& other)

{

Swap head

Swap tail

Swap size

}

//merge

bool merge(const Map& m1, const Map& m2, Map& result)

{

Create a new Map to store result, to avoid aliasing

Create Boolean variable to check if there are different values assigned to the same key

Repeatedly

Get the information ith largest element from m1

Check if it’s presented in m2

If yes, check for conflicts

If no conflicts, insert it to the new Map

Repeatedly

Get the information ith largest element from m2

Check if it’s presented in m1

If the element only exists in m2

Insert it to the new Map

Copy the new Map to Map result

}

//reassign

void reassign(const Map& m, Map& result)

{

Create a new Map to store result, to avoid aliasing

If there is only one node in the list

Directly copy it

Get key/value of head

Repeatedly

For every two consecutive nodes beginning at head

Exchange the value of the two nodes

Insert the two new nodes to the new Map

Get key/value of tail

Exchange the value of head and tail

Copy the new Map to Map result

}

Test Cases

1. My custom testing codes (Passed all assertions and got expected outputs for merge and reassign)

#include <iostream>

#include <cstdlib>

#include <type\_traits>

#include <cassert>

#include "Map.h"

using namespace std;

const KeyType ARRAY1[3] = {

"Fred", "Ethel", "Lucy"

};

const ValueType ARRAYV1[3] = {

123, 456, 789

};

const KeyType ARRAY2[2] = {

"Lucy", "Ricky"

};

const ValueType ARRAYV2[2] = {

650, 321

};

int main()

{

Map m00;

m00.insert(ARRAY1[0], ARRAYV1[0]);

m00.insert(ARRAY1[1], ARRAYV1[1]);

assert(m00.size() == 2); //After inserting, m00 should have 2 nodes.

Map m1;

assert(m1.empty()); //m1 is now an empty list

m1.swap(m00);

assert(m1.size() == 2);

assert(m00.size() == 0); //After swapping, m1 now should have 2 nodes and m00 has 0.

m1.insert(ARRAY1[2], ARRAYV1[2]);

assert(!m1.erase("RRR")); //Cannot erase something that does not exist in the list

Map m2;

m2.insert(ARRAY2[0], ARRAYV2[0]);

m2.insert(ARRAY2[1], ARRAYV2[1]);

KeyType k;

ValueType v;

m1.get(0, k, v); // Get Ethel's information

assert((k == "Ethel") && (v == 456));

m1.get(2, k, v); // Get Lucy's information

assert(v == 789); // In m1, Lucy's value is 789

m2.get(k, v); // Now get Lucy's information from m2

assert(v == 650); // In m1, Lucy has a different value;

assert(m2.update(k, 654)); // Check if update function is working

assert(m2.contains("Ricky")); // Check if contains function is working

Map result1;

merge(m1, m2, result1);

KeyType k1;

ValueType v1;

for (int i = 0; i < result1.size(); i++)

{

result1.get(i, k1, v1);

cout << k1 << " " << v1 << endl;

// Expected output: "Fred" 123 "Ricky" 321 "Ethel" 456

// might have differnet orders

}

cout << endl;

Map result2;

Map m3;

m3.insert("Fred", 123);

m3.insert("Ethel", 456);

m3.insert("Lucy", 789);

m3.insert("Ricky", 321);

reassign(m3, result2);

KeyType k2;

ValueType v2;

for (int i = 0; i < result2.size(); i++)

{

result2.get(i, k2, v2);

cout << k2 << " " << v2 << endl;

// Expected output: "Fred" 456 "Ethel" 123 "Lucy" 321 "Ricky" 789

// might have different orders

}

}

2. Also passed most cases in the provided tester, except for case 72, 74, 76, 77, and 84 which contain some specific homework 1 implementations.