Project 2 Report

1. The implementation of my doubly-linked list consists of a dummy node followed by other nodes, each containing a key, value, and a next and previous pointer. The linked list is circular, meaning when the map is empty, it just contains a dummy node with the next and previous pointers both pointing to the same dummy node. When a new node is inserted, it is inserted in the front, right after the dummy node.
2. bool Map::insert(const KeyType& key, const ValueType& value)

If the map doesn’t contain key,

Create a new node with given key and value

If the list is empty,

Assign the new node’s next/prev to dummy

Assign the dummy node’s next/prev to the new node

Else,

Assign new node’s prev to dummy

Assign new node’s next to original first node

Assign dummy’s next to new node

Assign the second node’s prev to new node

Increment size by one and return true

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

If the map doesn’t contain the key,

Return false

Set finder pointer to the first node

Search through the map until finder points to the dummy

If the node’s key equals the parameter key,

Set that node’s value to parameter value

Return true

bool Map::erase(const KeyType& key)

If the map contains one node,

Point next/prev back to dummy and delete the node

Else,

Search through the loop for the node corresponding to parameter key

Set the previous node’s next to the next node

Set the next node’s prev to the previous node

Delete the node corresponding to parameter key

Decrement size and return true

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

Set finder pointer to the first node

Loop through the map until finder = dummy

If the node’s key equals the parameter key

Return true

Return false if gets here

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

If the map does not contain key,

Return false

Loop through the map with finder like before

If node’s key = parameter key,

Break;

Set value parameter to the node’s value and return true

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

If i is out of range,

Return false

Set finder to the first node

Set finder to the next node i times

Set parameter key and value to finder node’s key and value

Return true

void Map::swap(Map& other)

Create a temp head and temp size from this

Assign this.head and this.m\_size to other’s head and size

Assign other’s head and size the two temp variables

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

Give result m1’s nodes with assignment operator

Loop through m2

If m1 does not contain m2’s node’s key,

Insert that key and value into temp

Else

If the values for m1 and m2’s corresponding key are different,

Set return value to false

Return the proper return value

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

Give result m’s nodes using assignment operator

If m is empty,

return

Record the first node’s key and value

Until it gets to the second-to-last node, loop through

Update each node’s value to the value of the next node

Set the last node’s value to the first node’s value

Set result equal to temp

1. These test cases should be run one at a time (after the first comment) with KeyType set to std::string and ValueType set to double. There should be for loops at the end that get() the key and value of each node in each map to ensure proper execution.

Map empty;

Map one;

Map copy;

one.insert("Juan", 5);

Map m(one);

m.insert("Adam", 29);

m.insert("Beth", 30);

m.insert("Colin", 12);

copy = m;

KeyType k;

ValueType v = 83;

//sets up the maps we'll be testing (also tests constructors/assignment operator/insert)

empty.empty(); //tests true case for empty

one.empty(); //tests false case for empty

empty.size();

one.size();

m.size(); //tests size for 0, 1, and 3 nodes

m.update("Adam", 26); //tests update

empty.update("Chris", 15); //tests update when key is not in map

m.insertOrUpdate("Aaron", 12); //tests insert case

m.insertOrUpdate("Adam", 42); //tests update case

empty.erase("Mary"); //nothing to erase

m.erase("Juan"); //tests erase for last node

one.erase("Juan"); //tests erase for only node

empty.contains("Hi"); //contains nothing

m.contains("Adam"); //tests true case

m.get("Beth", v); //makes sure v is changed by get

empty.get("Beth", v); //tests when map is empty

m.get(-2, k, v); //tests when i is out of range

m.get(0, k, v); //tests getting the first item

m.get(3, k, v); //tests getting the last item

m.swap(empty); //tests where swapping with empty

m.swap(empty); //swaps back

m.swap(m); //tests where swapping with self

one.insert("Juan", 68);

combine(one, m, empty); //tests combine where result is empty

combine(one, m, copy); //tests combine where result is empty/alias

reassign(one, empty); //tests case with one node

reassign(m, m); //tests case where both parameters are same