1. My design is a doubly-linked list of Node structs that contain next and previous pointers as well as a KeyType and a ValueType. It is not a circular design, so the prev pointer of the first item is NULL and the next pointer of the last item is also NULL. There is no dummy node. The nodes are in a particular order. Every time the insert function is called, a new node is inserted onto the end of the current list. The list ends up being chronologically ordered.
2. bool Map::erase(const Keytype& key)

if key isn’t in the list or list is empty

return false

if there is only one item in the list

delete the item, set head and tail pointers to null

return true

Loop through list to find item with the corresponding key

If the key that the looping pointer points to equals the key of the parameter

If the pointer points to the first item in the list

Set head equal to the next item

` Delete the first item

Set the previous pointer of the new head to the nullptr

If the pointer points to the last item in the list

Set tail equal to the second to last item in the list

Delete the last item

Set the next pointer of the new tail to the nullptr

If the pointer points to any item in the middle of the list

Set the next pointer of the previous item to the next item

Set the previous pointer of the next item to the previous item

Delete the Node that the looping pointer points to

Decrement the m\_size variable

Return true

Return false

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

If the key is already in the list

Return false

Create a new node

Set its key and value equal to the parameter key and value

Set the previous pointer of the new node to NULL

Set the next pointer of the new node to NULL

If the list is empty

Set head pointer to the newly created node

Set tail pointer to the newly created node

If the list isn’t empty (else)

Set previous pointer of the new node to the tail

Point the next pointer of the tail to the newly added Node

Set the tail equal to the newly added Node

Increment the size counter of the list

Return true

1. void Map::swap(Map& other)

Create two temporary pointers and set them equal to the head and tail pointers

Set head equal to the other’s head

Set tail equal to the other’s tail

Set the other’s head equal to the temporary pointer that holds the head’s old value

Set the other’s tail equal to the temporary pointer that holds the tail’s old value

Make a temporary integer variable that holds the current size of the current list

Set m\_size equal to the other’s size

Set the other’s size equal to the temporary variable that holds the size of current list

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

Create two temporary variables that copy the items in m1 and m2

While the result Map is not empty

Find the first item of the list

Delete the first item of the list

Repeat until result is empty

Declare a Boolean statement that checks if the values of the same key are equal

Loop through each item of m1

Store the keys and values of the ith item in m1 into two variables

If m2 does not contain the key found in m1

Insert the pair of values into the result Map

Otherwise, (if m2 does contain a key found in m1)

Get the value that corresponds with the key found in both lists

If the values of the key match

Insert the pair of values into the result

Otherwise, if the values do not match

Set the Boolean declared earlier to false, because vals don’t match

Loop through each item of m2

Store the keys and values of the jth item in m2 into two variables

If key of m2 does not show up in m1 nor the result map

Insert it into result map

Return the Boolean statement declared earlier that checks for unequal values of same key

1. void subtract(const Map& m1, const Map& m2, Map& result)

Create two temporary variables that copy the items in m1 and m2

While the result Map is not empty

Find the first item of the list

Delete the first item of the list

Repeat until result is empty

Loop from 0 up to the size of m1

Store the keys and values into two variables

If the current key of m1 does not appear in m2

Insert the pair of values into the result Map

1. Test cases:

// Map(), empty(), and operator=(map) tests

Map m;

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

m.insert("A", 1);

// Map(other) test

Map m1(m);

assert(!m1.empty()); // test for not empty cases

assert(m1.contains("A")); // test if item gets copied over

// insert(k) tests

m1.insert("B", 2);

m1.insert("C", 3);

m.swap(m1);

// size() tests

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

assert(m1.size() == 1); // test size

// ~Map() test

Map \*newguy = new Map;

newguy->insert("delete me", 42);

assert(newguy->size() == 1); // test size

delete newguy;

// update(k, v) test

m.update("A", 11);

ValueType mV;

assert(m.get("A", mV) && mV == 11); // test get and value

// insertOrUpdate(k, v) test

m.insertOrUpdate("B", 22); // test if value gets updated

m.insertOrUpdate("C", 33); // test if value gets updated

m.insertOrUpdate("D", 44); // test if value gets inserted

assert(m.get("B", mV) && mV == 22); // test get and value

assert(m.get("C", mV) && mV == 33); // test get and value

assert(m.size() == 4 && m.get("D", mV) && mV == 44); // test size and get and value

// erase(k), contains(k), and get(k, v) test

m.erase("B"); // test erase

assert(!m.erase(“V”)); // nothing to erase

assert(!m.get("B", mV)); // test get if key isn't in list

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

m.erase("D"); // test erase

assert(!m.contains("B") && !m.contains("D")); // test contains if key isn't in list

assert(m.size() == 2); // test if size updated after the erases

// get(i, k, v) test

m.insert("is why", 3);

m.insert("I live in", 4);

m.insert("a plaza", 5);

KeyType k;

ValueType v;

assert(m.get(0, k, v) && k == "A" && v == 11); // tests if every item in the list is there

assert(m.get(1, k, v) && k == "C" && v == 33);

assert(m.get(2, k, v) && k == "is why" && v == 3);

assert(m.get(3, k, v) && k == "I live in" && v == 4);

assert(m.get(4, k, v) && k == "a plaza" && v == 5);

assert(!m.get(5, k, v)); // test get if no key exists at the given i value

// swap(other) test

Map n;

n.insert("That", 1);

n.insert("is why", 3);

n.insert("you are anti-social", 4);

m.swap(n);

assert(m.size() == 3 && n.size() == 5); // test to see if swap also swapped the sizes

assert(m.get(2, k, v) && k == "you are anti-social"); // tests if a value in n appears in m after

// swap

// combine(m, n, result) test

Map result;

combine(m, n, result);

assert(result.size() == 7); // tests combine

m.insert("A", 11);

combine(m, n, result);

assert(result.size() == 7); // tests combine after inserting another key

// subtract(m, n, result) test

subtract(n, m, result);

assert(!result.contains("A") && !result.contains("is why")); // tests if subtract gets rid of same keys

assert(result.size() == 3); // tests to see if size of result is 3; n.size() - # of repeats (5 - 2 = 3)

assert(m.size() == 4 && n.size() == 5); // tests to see if sizes of m and n are unchanged after

// subtract

subtract(m, n, result);

assert(result.size() == 2); // tests if size is correct: m.size() - # of repeats (4 - 2 = 2)