Shreya Raman  
004923456

1. My doubly-linked list is linear with no dummy node. Each node has a mapStruct, a next pointer, and a prev pointer. The mapStruct holds variables KeyType and ValueType. New nodes are added to the bottom of the list.

Empty Map

nullptr

tail

head

tail

head

mapStruct

prev

mapStruct

next

mapStruct

next

prev

nullptr

prev

next

Regular Map

nullptr

1. Pseudocode

**Default Constructor**

Set size to 0

Set head/tail to nullptr

**Destructor**

For every node

Set a temporary node to current node

Set current node to next node

Delete temporary node

**Copy Constructor**

Initialize size and head/tail pointers

For every node

Get and insert node values to current Map

**Assignment Operator**

If parameter map is not equal to current map

Create identical temporary Map  
 Swap current Map with temp Map

Do nothing

**bool insert()**

If already contains key

Return false

Create new node and assign values

If empty list

Assign head/tail to node

Add node to beginning of map

Return true

**bool update()**

For every node

If a node’s key equals the key parameter

Change parameter value to node value

Return true

Return false

**bool insertOrUpdate()**

If Map can be updated

Update and return true

Insert new node and return true

**bool erase()**

If key is not in Map or list is empty

Return false

If only one node

Delete node, decrement size

If target is head or tail

Delete node and repoint next and prev pointers

If target is in middle

Set next pointer of previous node to that of after

Set prev pointer of following node to that of before

Return false

**bool contains()**

For every node

If node key matches parameter

Return true

Return false

**bool get()**

For every node

If node key matches parameter

Change value parameter to that of node

Return true

Return false

**bool get()**

If i is out of bounds

Return false

Traverse linked list to i

Change key and value

Return true

**void swap()**

Make temp variables for head, tail and size

Swap head pointers

Swap tail pointers

Swap linked list sizes

**void combine()**

Make temp Maps of parameters m1 and m2

Make a temp bool set to true

Swap a temp Map with result

Go through every value of other temp2 map

Get values of temp2

If temp does not have that node

Add node to result

If temp has node but different values

Erase node from result

Set bool to false

Return bool

**void subtract()**

Make temp Maps of parameters m1 and m2

Construct empty map

Swap empty map with result

For every value of temp map

Get values of temp map

If other temp map does not have node

Insert node to result map

1. Test Cases:

    Map m;                      // Default constructor

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

    assert(!m.erase("hello")); // Tests erase

    m.insert("A", 1.23); // Tests insert function

    m.insert("B", 4.56);

    m.insert("C", 7.89);

    assert(m.size() == 3);      // Tests size function

    KeyType k = "B";

    ValueType v = 9.87;

    KeyType k1 = "D";

    ValueType v1 = 6.54;

    KeyType k2 = "A";

    ValueType v2 = 3.21;

    KeyType k3 = "B";

    ValueType v4 = 0.12;

    ValueType v3;

    KeyType key;

    ValueType val;

    assert(m.get(0, key, val) && key == "A" && val == 1.23); // Tests get function

    assert(!m.get(-1, key, val) && key == "A" && val == 1.23); // Tests invalid i in get function

    assert(m.contains("A") && m.contains("B") && !m.contains("D"));

    assert(m.update(k, v) && m.get(k, v3) && v3 == 9.87); // Tests update function

    assert(!m.insert(k3, v4) && m.get(k3, v3) && v3 == 9.87); // Tests invalid insert

    assert(m.insertOrUpdate(k2, v2) && m.get(k2, v3) && v3 == 3.21); // Tests insertOrUpdate function

    assert(m.size() == 3);

    assert(m.insertOrUpdate(k1, v1) && m.get(k1, v3) && v3 == 6.54);

    assert(m.erase("C") && m.size() == 3); // Tests erase function

    assert(!m.get("C", v3)); //Tests invalid get function

    assert(v3 == 6.54 && !m.contains("C"));

    // Tests swap function

    Map m1;

    m1.insert("Y", 9.87);

    m1.insert("Z", 1.23);

    m.swap(m1);

    assert(m.contains("Z") && m.contains("Y"));

    assert(m1.contains("A") && m1.size() == 3);

    // Tests assignment operator and copy constructor

    ValueType vv;

    Map m2(m1); // Tests copy constructor

    assert(m2.contains("A") && m2.contains("B") && m2.get("B", vv) && vv == 9.87 && m2.size() == 3);

    Map m3;

    m3.insert("S", 1.23);

    m3 = m1; // Tests assignment operator

    assert(m3.contains("D") && m3.contains("B") && m3.get("D", vv) && vv == 6.54 && m3.size() == 3);

    assert(m3.erase("A") && m3.size() == 2); // Tests erasing head

    m3 = m2;

    assert(m3.erase("B") && !m3.contains("B")); // Tests erasing middle

    m3 = m1;

    assert(m3.erase("D") && m3.size() == 2); // Tests erasing tail

    // Tests combine and subtract functions

    Map map;

    map.insert("a", 0);

    map.insert("b", 1);

    map.insert("c", 2);

    map.insert("d", 3);

    map.insert("e", 4);

    Map map4(map);

    Map s(map);

    Map map1;

    map1.insert("d", 3);

    map1.insert("e", 6);

    map1.insert("f", 7);

    Map r(map1);

    Map result;

    assert(!combine(map, map1, result) && result.size() == 5 && !result.contains("e") && result.contains("d"));

    assert(!combine(map, map1, map) && map.size() == 5 && !map.contains("e") && map.contains("d"));

    Map map2;

    map2.insert("h", 7);

    map2.insert("g", 8);

    assert(combine(map4, map2, result) && result.size() == 7 && !result.contains("f"));

    Map w;

    w.insert("a", 1);

    w.insert("b", 2);

    w.insert("c", 3);

    w.insert("d", 4);

    w.insert("e", 5);

    Map x;

    x.insert("a", 5);

    x.insert("b", 4);

    x.insert("c", 6);

    x.insert("d", 2);

    x.insert("e", 1);

    Map i;

    assert(!combine(w, x, i)); // Checks if none are combined

    assert(i.empty());

    Map results;

    subtract(s, r, results); // Tests subtract

    assert(results.size() == 3 && results.contains("a") && !results.contains("d") && !results.contains("f"));

    subtract(s, r, s);

    assert(s.size() == 3 && s.contains("a") && !s.contains("d") && !s.contains("f"));

    Map f;

    f.insert("a", 1);

    f.insert("b", 2);

    f.insert("c", 3);

    f.insert("d", 4);

    f.insert("e", 5);

    Map g;

    g.insert("a", 5);

    g.insert("b", 4);

    g.insert("c", 6);

    g.insert("d", 2);

    g.insert("e", 1);

    Map e;

    subtract(f, g, e); //Checks if all are subtracted

    assert(e.empty());