1. The linked list has two elements to it i.e. the key and the value. Apart from that there are two pointers: next and prev that point to the next and previous nodes respectively. The head and tail pointers point to the start and end of the list. An empty map has no nodes with the head and tail both pointing to nullptr.
2. ***Constructor***

Set head and tail to nullptr

Set size to 0

***Empty***

Check if head and tail are both nullptr and size is 0

Return true

Else

Return false

***Insert***

If key already exists

Return false

Create new node q

Set value of q and key of q to parameters

If head and tail are null

Set head and tail to q

Set prev and next to null

Else

Set the next pointer of tail point to q

Set prev pointer of q to tail

And next pointer of q to null

Set tail to q

Increase size and return true

***Update***

----repeatedly----

Go through the linked list using p

If matching key is found

Set value of that key to parameter sent

Return true

----repeatedly---

return false

***insertOrUpdate***

----repeatedly----

Go through the linked list using p

If matching key is found

Set value of that key to parameter sent

Return true

----repeatedly---

Create new node q

Set value of q and key of q to parameters

If head and tail are null

Set head and tail to q

Set prev and next to null

Else

Set the next pointer of tail point to q

Set prev pointer of q to tail

And next pointer of q to null

Set tail to q

Increase size and return true

Contains

---repeatedly---

navigate through list using p

is key of the parameter found in linked list

return true

---repeatedly---

return false

***erase***

---repesatedly---

use p to navigate through list to check if any matching key exists

set t to 1 and break

---repeatedly---

if no key was found return false

else if the found key was the only element in the list

delete that element

decrease size and return true

else if it was the first element

set head to the next address it points to

delete that pointer and reduce size

set the prev of the new head to null

return true

else if it is the last element

set tail to point to prev pointer address

delete that pointer

decrease the size and set next pointer to null

return true

else

set the next of the prev of the pointer to the next of that pointer

set prev of the next of the node to point to the prev of the pointer

delete the pointer and decrease the size

return true

get

---repeatedly---

go through the list using p to search for key

if found

set value to the value of the node and return true

---repeatedly---

return false

get (3 parameters)

if i is out of range or size is 0

return false

repeatedly move to the next node i times

set value and key of the parameters to those contained in the node

return true

***copy constructor***

set size of new object to current size

make head a new node

set values of head’s prev to nullptr

---releatedly---

traverse through the existing map till last element

set the value and key of the existing map’s object to the new node

while it doesn’t reach last node of existing map

set next pointer of each new node to a new address

set prev pointer of next of each new node to equal q(new node)

increment q to point to next node

---repeatedly---

when reaches last node of existing object

set pointer pointing to new object’s next to null

set tail to new object’s pointer q

***destructor***

traverse through the list using n

create m to store address of next node

delete n

set n to equal address of next node

***operator***=

traverse through the list using n

create m to store address of next node

delete n

set n to equal address of next node

set size of new object to current size

make head a new node

set values of head’s prev to nullptr

---releatedly---

traverse through the existing map till last element

set the value and key of the existing map’s object to the new node

while it doesn’t reach last node of existing map

set next pointer of each new node to a new address

set prev pointer of next of each new node to equal q(new node)

increment q to point to next node

---repeatedly---

when reaches last node of existing object

set pointer pointing to new object’s next to null

set tail to new object’s pointer q

***swap***

swap the sizes of the two maps

swap the heads of the two maps

swap the tails of the two maps

***combine***

create temp map variable

---repeatedly---

go through first linked list m1 to insert all elements into temp

go through second linked list m2

if any existing key is found

check if values are same

if same then do nothing

else

set t=1 to return false later

erase the node containing key

set result to temp

if t is 1

return false

else

return true

***subtract***

temp map variable

traverse through linked list 1

---repeatedly---

get values of key and value of each node

check if other map m2 contains that node with that key

---repeatedly---

set result to temp

3.

int main()

{

Map a; //constructor

assert(a.size()==0);// check size

assert(!a.erase("ruby")); //nothing to erase

assert(a.empty()); //check if empty

assert(a.insert("Rob", 123) && a.size()==1 && !a.empty()); //checking insert and if size is increased

assert(a.insert("Bob", 234));

Map b=a; //using copy constructor

assert(b.size()==2 && b.contains("Bob") && b.contains("Rob")); //checking if both elements were added

assert(!b.update("Cob",100));// checking of update compiles

assert(b.insertOrUpdate("Rob",100) );

ValueType v;

assert(b.get("Rob",v) && v==100 && v!=123);//checking if update does its job

KeyType k;

assert(!b.get(4, k, v)); //I is greater than size

assert(b.get(1, k, v) && b.contains(k)); //checking if get returns a key that exists

Map c;

c.operator=(b);

assert(c.size()==2 && c.contains("Rob")); //checking if assignment operator works

assert(c.erase("Rob") && !c.erase("Beth")); //erases existing rob and returns false for non existing beth

c.swap(a);

assert(c.size()==2 && a.size()==1 && a.contains("Bob") && c.contains("Rob") ); //checking if swap happened

Map d;

assert( d.insert("Lucy" ,12) && d.insert("fred", 13) && d.insert("Ethel" ,14) && !d.insert("Lucy", 18));

assert(combine(c, d, b) && b.size()==5 && b.contains("Lucy") && b.contains("Bob")); //combine sets b to equal to combination of the other two

assert(c.insert("Lucy", 13) && !combine(c, d, a) && a.size()==4); //c is a combination of all elements of the two maps except lucy as they had different values and returns false

assert(c.update("Lucy", 12));

subtract(c, d, a);

assert( a.size()==2 && a.contains("Rob") && a.contains("Bob")); //b contains all elements of a not in d

cout<<"Passed All Tests"<<endl;

}