1. I do have a dummy node for most of my functions, some of them having one and some of them having two. My dummy node if usually assigned the head address so it can go through the list, while the head always points to the first node. The dummy node checks through the list to usually look for a specific key or if some key is in the list, so we know whether we can add a key, update that key, or simply know if the list contains that key. Nodes are appended to the end of the list. In each of the nodes, there is a key that corresponds to a specific value, there is a pointer that points to the next node in the list, and a pointer that points to the previous node in the list. The list is not circular.
2. …constructor

Initializes head pointer to null

Num is set to 0

…

…empty

If num is 0

True

Else , false

…

… size

Return size of list

…

… insert

Initialize a pointer to pair variable

Initialize a pointer to pair variable and set it to null

If empty

Initialize a pointer variable, node, to a Pair struct

the value node points to is set to value parameter

the key node points to is set to key parameter

the m-next pointer is set to the head

head is set to node

num is incremented by 1

return true

else

repeatedly…

if the m\_next is null

Initialize a pointer variable, node, to a Pair struct

the value node points to is set to value parameter

the key node points to is set to key parameter

node’s m\_next is set to curr m\_next

node’s m\_prev is set to curr

curr’s m\_next is set to node

curr’s m\_prev is set to prev

num is incremented by 1

returns true

return false

…

… update

Initialize a pointer, p, to a Pair

Repeatedly…

If the pointer p is not null

The value p points to is set to the value in the parameter

Return true

Else , return false

…

…insert or Update

Initialize a pointer, curr, to a Pair

Initialize a pointer, prev, to a Pair and set it to null

If empty

Initialize a pointer variable, node, to a Pair struct

the value node points to is set to value parameter

the key node points to is set to key parameter

the m-next pointer is set to the head

head is set to node

num is incremented by 1

return true

else

repeatedly…

if the curr is not null

the value current points to is set to the value in the parameter

else if the m\_next is null

Initialize a pointer variable, node, to a Pair struct

the value node points to is set to value parameter

the key node points to is set to key parameter

node’s m\_next is set to curr m\_next

node’s m\_prev is set to curr

curr’s m\_next is set to node

curr’s m\_prev is set to prev

num is incremented by 1

return true

…

…erase

Initialize a pointer, p, to a Pair

Repeatedly;

If p is not null

If m\_prev is nunll

Head is set to the m\_next p points to

Else if m\_next is null

M\_next of the previous node p points to is set to null

Else

M\_next of the previous node p points to is set to m\_prev

M\_prev of the next node p points to is set to m\_next

Delete p

Num is decremented by 1

Return true

Return false

…

…contains

Initialize a pointer, p, to a Pair

Repeatedly…

If p is null

Return false

Return true

…

…get

Initialize a pointer, p, to a Pair

Repeatedly…

If p is not null

Value is set to the value p points to

Return true

Return false

…

…get with parameter i

Initialize a pointer, p, to a Pair

If I is greater than 0 and less than the size of the list

If list is empty

Return false

Repeatedly…

P is set to the m\_next until the position is at the parameter i

Key is set to the key p points to

Value is set to the value p points to

Return true

Return false

…

…swap

Initialize a pointer to a Pair, temp, and set it to the head

Head is set to the other map’s head

The other map’s head is set to temp

Initialize an int, t, and set it to num

Num is set to the other map’s num

The other map’s num is set to the t

…

…copy constructor

Initialize num to 0

Initialize head to nullptr

A pointer to Pair struct is set to the other head

Repeatedly…

inserts each value from the other list to this list

increment sum to point to the next pointer

…

… assignment operator

If the two objects are different

A pointer to Pair struct is set to head

Repeatedly…

A node is deleted

Initialize num to 0

Initialize head to nullptr

A pointer to Pair struct is set to the other head

Repeatedly…

inserts each value from the other list to this list

increment sum to point to the next pointer

return the address

…

… destructor

A pointer to Pair struct is set to head

Repeatedly…

A node is deleted

…

…combine

Initialize a string z

Initialize a double z1

Initialize a boolean, answer to true

If result does not point to the same list as m1 or m2

While result is not empty

Get each node at the index

Erase that node

Repeatedly…

Intialize string x

Initialize double y1 and y2

If m1 gets a pair at the index

If m2 does not have that key or has the same pair

Result inserts it to its list

Else

Answer is set to false

Repeatedly…

Intialize string x

Initialize double y1 and y2

If m2 gets a pair at the index

If m1 does not have that key or has the same pair

Result inserts it to its list

Else

Answer is set to false

Return answer

…

…subtract

Initialize a string z

Initialize a double z1

If result and m1 are aliases

Create a temp map with copy constructor

Set result to temp map

While result is not empty

Get the value at the index 0

Erase each node from result

If result and m2 are aliases

Create a temp map with copy constructor

Set result to temp map

While result is not empty

Get the value at the index 0

Erase each node from result

If m1 and m2 are aliases

While result is not empty

Get the value at the index 0

Erase each node from result

Else

While result is not empty

Get the value at the index 0

Erase each node from result

Repeatedly…

Initialize string x

Initialize double y1 and y2

If m1 gets a pair at the index

If m2 does not have that same pair

Result inserts it to its list

…

1. Map m;//uses default constructor to make and empty map

assert(m.insert("Fred", 123)); // key and value were inserted

assert(m.insert("Ethel", 456));

assert(m.size() == 2); //checks if the size of the list is two

assert(m.contains("Fred"));//checks that Fred should be in the list

ValueType v = 42;

assert(!m.get("Lucy", v) && v == 42); //Lucy should not be in the list and v should be 42 still

assert(m.get("Fred", v) && v == 123); //v should be changed to 123

v = 42;

KeyType x = "Lucy";

assert(m.get(0, x, v) &&

((x == "Fred" && v == 123) || (x == "Ethel" && v == 456))); //one of those pairs should be copied into x and v

KeyType x2 = "Ricky";

assert(m.get(1, x2, v) && //the same value pair should not be copied into x2 and v

((x2 == "Fred" && v == 123) || (x2 == "Ethel" && v == 456)) &&

x != x2);

assert(m.insert("Lego", 003));

assert(m.insertOrUpdate("Fred", 32)); //updates fred since it is in the list

assert(m.insertOrUpdate("Lego", 7));

assert(m.erase(“Lego”)); //erases a node from the end of the list

assert(m.erase(“Fred”)); //erases a node from the beginning of the list

assert(m.erase(“Ethel”)); //erases a node from the middle of the list

Map k;

assert(k.insert("Craig", 556)); // creates a new map

assert(k.insert("Louise", 34));

assert(k.insert("Fred", 32));

m.swap(k); //swaps this map with the previous map

for (int i = 0; i < k.size(); i++) //checks if the maps correctly swapped

{

string x;

double y;

k.get(i, x, y);

cout << "Name: " << x << " Number: " << y << endl;

}

Map hello;

hello.insert("Maybe", 13); //checks if the combine function will erase

combine(m, k, hello); //combines both the lists

for (int i = 0; i < 6; i++) //checks if the lists correctly combined

{

string x;

double y;

hello.get(i, x, y);

cout << "Name: " << x << " Number: " << y << endl;

}

cout << endl;

Map hello(m); //checks the combine in terms of aliasing

combine(m, k, hello);

for (int i = 0; i < 6; i++)

{

string x;

double y;

hello.get(i, x, y);

cout << "Name: " << x << " Number: " << y << endl;

}

cout << endl;

Map sub; //checks the subtract function

subtract(m, k, sub);

for (int i = 0; i < 6; i++) //to see whether the subtract functions properly

{

string x;

double y;

sub.get(i, x, y);

cout << "Name: " << x << " Number: " << y << endl;

}

Map sub;

sub = m;

subtract(m, k, sub); //checks the subtract function in terms of aliasing

for (int i = 0; i < 6; i++)

{

string x;

double y;

sub.get(i, x, y);

cout << "Name: " << x << " Number: " << y << endl;

}