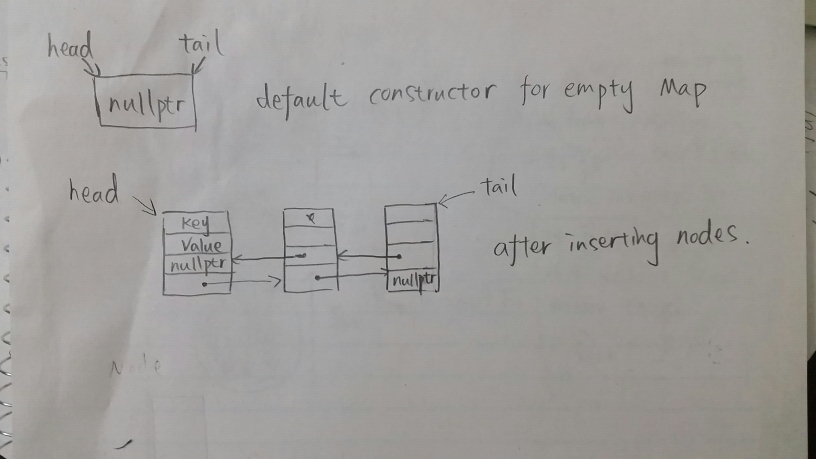
My design of the Map double-linked list comprises mainly of two structures, mapNode and dataPair. dataPair holds two values: a key (of KeyType) that is essentially connected to a value (of ValueType). On the other hand, each mapNode contains a dataPair and two pointers linking that mapNode to other mapNodes preceding and succeeding it.

The Map itself has a head pointer for the first mapNode in the list, and a tail pointer for the last. When new keys/values are inserted, a new mapNode is attached to the end by the tail, and that node becomes the new tail. Nodes are basically arranged from oldest-added items from the left to the most recently-added items on the right. The list I implemented is not circular, nor does it use a dummy node.

Structure for Map: doubly linked list



**Map()**

Initialize Map size to 0

Set head and tail pointers to NULL

**Map(const Map& other)**

Initialize Map size to 0 Set head and tail pointers to NULL For each mapNode in other:

Store the key and value of that mapNode into temporary values k, v Insert k, v into the current Map

**~Map()**

use temp node to hold the node to remove

repeatly until head is not equal to nullptr

head to next

delete node

set tail to nullptr

set count to zero

**Map& Map::operator=(const Map& rhs)**

Check aliasing

Copy rhs to temp

Swap temp with this object

Return this object

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

check if the Map is empty

create a new node and insert key and value

let head and tail point to the same node

return true

else if the Map contains the key

create a new node and insert key and value

set new node points to tail

tail points to new node

increment count

return true

if contains the key

return false

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

if the Map is empty and not contains the key

return false

else

create a temp node pointer

use while loop to update temp to the key node

set value to that node

return true

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

if the Map can update or insert

return true

else

return false

**bool Map::erase(const KeyType &key)**

if the Map not empty and contains the node having that key

if Map only have one node

delete that node

decrement counter;

otherwise

create curr node

iterate current Map

if it first node

set head points to next node

delete curr node

if it is the last node to delete

set tail to prev

delete curr node

else

set the previous node to point to next node of current node

set the next node points to previous node

decrement counter

return true

else map is empty or does not contain key

return false

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

if m1 or m2 is empty

set result to m1 or m2

return true

create differentValue to remember if it is combine failure or succeed

create temp1 and temp2 maps to hold m1 and m2 to prevent aliasing

clear result

use two for loops repeatly compare keys and values of the first Map and second Map

create bool found and set it to false to control the inner for loop

if they have the same key and value

erase node from m2

insert into result

set found to true

if they have the same key not the same value

erase node from m2

set found to true

set differentValue to true

if found is false

insert node into result

use for loop to insert the rest distinguished nodes into result

**void subtract(const Map& m1, const Map& m2, Map& result)**

if m1 is empty nothing to subtract from

set result to m1

return

create temp1 and temp2 maps to hold m1 and m2 to prevent aliasing

clear result

repeatly check nodes in temp1

if temp2 contains the node have the same key as in temp1

check next node

else temp1 do not have the same key

insert that into result