### Linked List Implementation

I chose to implement a simple doubly-linked list by

1. using a head pointer to point to the first node, rather than a dummy node
2. not implementing a tail node; the last node points to nullptr
3. having a linear list (non circular).

I chose these features for simplicity so that I’d be able to work with a very fundamental understanding of linked-lists. In addition, I added a private member variable m\_numElements to keep track of the size of the list.

Each node is a struct containing public members

1. A key and value of KeyType and ValueType types, respectively
2. a pointer to the next node, and
3. a pointer to the previous node.

Although nodes are in no particular order, when they are inserted, they are inserted at the beginning (like a stack: last in, first out). This makes the insert function not have to deal with special edge cases.

### Pseudocode

bool erase(const KeyType& key);

check if map contains node w/ given key

while not at end of list, repeatedly check each node:

if key in node matches given key

if deleting first node in list

set head to nullptr

if deleting last node

link second to last node to nullptr

if deleting from middle

set previous node’s next to deleted node’s next

set deleted node’s previous to deleted node’s previous

delete node

return true

return false if no node found

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

swap copy of m1 with result

iterate through each node of m2

extract each key and value from m2

if result contains key

remove node from result

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

swap copy of m1 with result

iterate through each node of m2

extract each key and value from m2

if result contains key

compare values from result and m2 for that key

if not equal

indicate to return false after loop (1)

delete node from result

otherwise

insert node into result

return T or F based on (1)

### List of Test Cases

Basic Test Cases

Map m;

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

assert(m.size() == 0); // test size empty

assert(m.insert("Fred", 123));

assert(m.insert("Pizza", 234));

assert(m.size() == 2); // test size

assert(m.update("Pizza", 125) && !m.update("Alissa", 2) && m.insert("mofo", 2018) && !m.insert("mofo", 123)); // test that updates and inserts work

m.insertOrUpdate("Alissa", 239);

m.insertOrUpdate("BBGURL", 69);

// test that insertOrUpdate function works

assert(m.contains("Alissa") && m.contains("BBGURL") && !m.contains("Star Wars"));

// test that erase works

assert(!m.erase("Star Wars") && m.erase("BBGURL") && !m.contains("BBGURL"));

double value;

m.get("Pizza", value);

assert(value == 125);

Testing for a larger map

Map yay;

assert(yay.empty());

std::string foods[26] = {

"Sushi", "Poke", "Coffee", "Beans", "Burgers", "Eggplant", "Ramen", "Hot Dog",

"Pasta", "Chicken", "Curry", "Tikka Masala", "Squash", "Boba", "Shrimp", "Chips",

"Taco", "Burrito", "French Fries", "Milkshakes", "Ice Cream", "Nachos", "Salmon",

"Tofu", "Pancakes", "Waffles"

};

for (int i = 0; i < 26; i++) {

yay.insert(foods[i], i \* i);

}

for (int i = 0; i < 26; i++) {

std::string key;

double value;

yay.get(i, key, value); // test get function

assert(key == foods[25 - i] && value == (25 - i)\*(25 - i)); // test get

}

Testing Swap functions

int yaySize = yay.size();

int mSize = m.size();

yay.swap(m);

assert(m.size() == yaySize && yay.size() == mSize); // test sizes swapped

for (int i = 0; i < 26; i++) {

std::string key;

double value;

m.get(i, key, value); // test each value got swapped

assert(key == foods[25 - i] && value == (25 - i) \* (25 - i));

}

assert(yay.contains("Fred")); // test value got swapped

m.swap(yay);

for (int i = 0; i < 26; i++) {

std::string key;

double value;

yay.get(i, key, value); // check each value got swapped back

assert(key == foods[25 - i] && value == (25 - i) \* (25 - i));

}

m.swap(m); // swap itself and test below

Map empty;

m.swap(empty); // check swap with an empty map

assert(m.size() == 0 && !m.contains("Pizza"));

Testing combine and subtract

Map m1, m2, result;

m1.insert("Fred", 123);

m1.insert("Ethel", 456);

m1.insert("Lucy", 789);

m2.insert("Lucy", 789);

m2.insert("Ricky", 321);

assert(combine(m1, m2, result)); // use the examples given in spec

assert(result.contains("Fred") && result.contains("Lucy"));

Map m3;

m3.insert("Ricky", 321);

m3.insert("Lucy", 789);

m3.insert("Ethel", 654);

subtract(m1, m2, result);