Tracy Yu

Project 2: (Double Trouble)

My design for the doubly-linked list is a circular linked-list that includes a dummy node. In each of the node, there are a pair of key and value and two pointers, next and prev. A head pointer always points at the dummy node to show the starting point of the list, which occurs at head->next or head->prev. This allows the user to never have to deal with the NULL pointer. The insertion of a new node occurs at the top of the list, after the dummy node, while deletion occurs wherever the key is found in the list.

An empty list with only a dummy node:

head

??????

A Doubly linked-list with two nodes and a dummy node:

head

??????

key: “Ethel”

pair: 123

key: “Fred”

pair: 345

Psuedocode:

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

assign result map m1

set temporary key and value

loop through m2 size

get each pair from m2

compare to key

then compare the value, if value is equal,

insert the pair and set true

else

erase the key

return false

if key does not exist in m1, insert the key into result

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

Set a temporary map equal to m1

Create temporary key and value

Loop through m2 size

get one of m2 value pairs

check if temporary map contains key

if it does, erase; else do nothing

result is equal to temporary map

Test cases:

Map p; // default constructor

// For an empty map:

assert(p.size() == 0); // test the size or number of pairs;

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

assert(p.insert(“Fred”, 1245)); // test insert

assert(p.insert(“Sean”, 9765)); // test insert

assert(p.update(“Fred”, 1000)); // test update function, should change value to 1000

double s = 10;

assert(p.get(“Fred”,s) && s == 10); // test the value is 10;

assert(p.insert(“John”, 1245)); // test insert

assert(p.erase(“Sean”)); // erase Sean from list

assert(p.size() == 2); // test the number of pairs

assert(!p.contains(“Jack”)); // does not contain Jack

Map q;

assert(q.insert(“Jill”, 1786)); // test insert

assert(q.insert(“Sean”, 9700)); // test insert

assert(q.insert(“Andrew”, 1245)); // test insert

q.swap( p); // test swap of q and a

string v;

int d;

assert(p.get( 1, v , d) && ( v == “Jill” && d == “1786” ) || ( v == “Sean” && d == “9765” ) ||( v == “Andrew” && d == “1245” ) ); // test the get function with different i value

string t;

assert(p.get( 0, t , d) && s != t && ( t == “Jill” && d == “1786” ) || ( t == “Sean” && d == “9765” ) ||( t == “Andrew” && d == “1245” ) ); // test the get function, and should get different values for t

Map n;

combine( q , p , n); // test combine function

assert(n.contains(“Sean”)); // test that Map n should not contain Sean

Map k = q; // test the copy constructor

p = q; // assignment operator

assert(p.contains("Fred")); // test if it copied q items

assert(p.contains("John")); // test if it copied q items

assert(!p.contains("Sean")); // test if it copied q items

Map result;

subtract( k , q , result); // test the subtract function

assert( result.empty()); //should be an empty list