Komal Prakash

CS32

Project 2

My doubly linked list consists of two dummy nodes. A head dummy node and a tail dummy node. The list is not circular. It is linear. The prev of the head points to null and the next of the tail points to null. Nodes/Elements (as I call them) are inserted to the end of the list. Each element consists of k for key, v for value, next pointer and a previous pointer.

nullptr

Empty Map: Not Empty Map:

nullptr

next

next

prev

prev

v

v

k

k

head

head

prev

k

First element of list

next

next

prev

v

v

k

tail

prev

k

nullptr

next

v

tail

nullptr

Pseudocode:

Erase:

traverse through the map (create separate search function) and output the element with the desired key

if the desired key exists in the map

point the previous element to the one after the one to be erased

point the next element to the one before the one to be erased

delete the target element

reduce the map size

return true

otherwise return false

Combine:

While the result in not empty, erase each element

Add everything from m1 into the result

For each element in m2, get the m2 key and value

Check if the key in m2 is in m1

if it is, check if the vals are equal

if it is, erase that element from result and return false

if key in m2 not in m1, insert it

Reassign:

While the result is not empty, erase each element

Return an empty result if m is empty

Get the key and value of the first and last element

Insert last element value into first element key

For all other elements other than the first element

Insert the previous element’s value into the next element

Test cases:

**int** main() {

    Map komal;

    assert(komal.empty()); //tests empty

    assert(komal.insert("prakash",50));//checks insert function, returns true if successfully added

    assert(!komal.insert("prakash",40));//insert should not add another element with existing key name and different value

    assert(komal.insert("nishant",21));//additional insert test

    assert(komal.insert("pp",345));//additional insert test

    assert(!komal.empty()); //tests empty again (should not be empty)

    assert(komal.size()==3); //tests size

    assert(komal.update("prakash",40)); //tests update

    assert(!komal.update("lalala",40));//tests update (should not update because no matching key)

    assert(komal.insertOrUpdate("prakash", 75));//tests insertOrUpdate (should update)

    assert(komal.insertOrUpdate("rainbow", 7)); //tests insertOrUpdate (should insert)

    assert(!komal.erase("joe")); //should not erase

    assert(komal.erase("rainbow")); //should erase rainbow element

    assert(komal.contains("prakash")); //tests contains

    assert(!komal.contains("rainbow")); //should not contain

**double** value=0;

    assert(komal.get("prakash",value)); //test get function

    assert(value==75); //checks if correct value inputted to value variable from get

    assert(!komal.get("dodo",value)); //checks get (no such key in Map)

    assert(value==75); //checks that value has not changed

    string key;

**double** num;

    assert(komal.get(0,key,num)); //tests 3 parameter get function

    assert(key=="prakash"&&num==75); //tests get (makes sure key and value are of the first element)

    komal.dump(); //print current komal Map

    cerr<<"-----copy constructor-----\n";

    Map a(komal); //tests copy constructor

    a.dump(); //prints a to check if copy constructor worked

    cerr<<"-----swap func-----\n";

    Map b;

    b.insert("hey",3);

    b.insert("joe",4);

    Map c(komal);

    Map d(b);

    c.swap(d); //tests swap

    d.dump(); //print d to see if it equals komal

    b.dump(); //print b to see if it equals c

    cerr<<"-----assignment operator-----\n";

    b=komal; //checks assignment operator

    b.dump(); //print b to see if it equals komal

    cerr<<"-----reassign-----\n";

    Map f;

    f.insert("hihihi",33);

    f.insert("yo",66);

    reassign(komal,f); //tests reassign

    f.dump(); //print f to see if it correctly emptied f and reassigned komal

    komal.dump(); //print komal to see if komal stayed the same

    Map empty;

    reassign(komal,empty); //tests reassign again with an empty second parameter

    empty.dump();//print empty to make sure it reassigned komal correctly

    Map jorge;

    jorge.insert("heyyyy",55);

    reassign(jorge,empty); //tests reassign

    empty.dump(); //print empty to make sure it looks just like jorge

    Map sandy;

    sandy.insert("whatsup", 45);

    empty.insert("prakash", 75);

    empty.insert("nishant",22);

    cerr<<"-----combine-----\n";

    cerr<<combine(komal,empty,sandy); //tests combine (should return 0 for false)

    sandy.dump(); //print sandy to make sure it emptied, correctly kept "prakash" and removed "nishant" entirely

    empty=komal; //make empty equal to komal

    cerr<<combine(empty,komal,sandy); //tests combine (should return true)

    sandy.dump(); //print sandy and check if sandy looks just like komal

    cerr<<"-----reassign again-----\n";

    reassign(komal,empty); //tests reassign

    cerr<<combine(komal,empty,sandy); //tests combine (should return false because of same keys, different values)

    assert(sandy.empty()); //tests if sandy is empty

}