**Report – Project 2**

Code Design:

1. Each node will have a previous (pointer to the node before it), a next (pointer to the node after it), a KeyType (with the value of the Key), and a ValueType (with the value of the Value). KeyType & ValueType will be stored as a struct.
2. The Map will also keep track of the beginning/end of the linked list using a pointer called head (pointing to first element) and tail (pointing to the last element).

Pseudocode:

1. Erase

//make sure it's in the map before iterating

//make sure list isn't empty

//check if there’s only 1 node

//check if item is in the front

//check if item is the end

//check if item is somewhere in the middle

//iterate until you reach item and delete item

1. Swap

//exchange the sizes

//create temp head/tail, set it to a temp Node

//set tail/head for the current map using other

//tail/head for other map using temp

1. Insert

//check if this value is already in the map, return false if it is

//check if this is our first time adding something to the linked list

//if yes then:

//set the "next" & "previous" values of the new node

//set the values of the new node

//set the values of the previous one

//map isn’t empty, append to end

//set the "next" & "previous" values of the new node

//set the values of the new node

//set the values of the previous one

//set tail

1. Get (2 Args)

//set new node to head & iterate until you reach the end of map size

//if iterator doesn’t point to null and keyvalue == key, set value equal to Keymap’s Value and return true

//otherwise return false

1. Get (3 Args)

//make sure that i is within range

//set new node to head & iterate until you reach the end of map size

//if iterator doesn’t point to null and keyvalue == key, set value equal to Keymap’s Value and return true

//otherwise return false

1. Update

//set new node to head & iterate until you reach the end of map size

//if iterator doesn’t point to null and keyvalue == key, use value to set Keymap’s Value and return true

//otherwise return false

1. Combine

//empty result if it’s not empty

//loop through map 1

//get the value of something in map1

//if it’s is also in m2 -> make sure it has different values or set flag to false, add to result

//it is not also in m2 -> add it to result

//loop through map2 and only add unique elements

To test the functions:

1. **Testing size function, empty function, contains function**

void test\_SizeEmptyContains(Map A)

{

//make sure Mapsize is 0 since empty

KeyType Key2 = "a";

KeyType Key3 = "b";

assert (A.contains(Key2)==false);

assert (A.contains(Key3)==false);

assert (A.size()==0);

assert (A.empty()==true);

//after inserting 1 element

A.insert(Key2, 3.4);

assert (A.contains(Key2)==true);

assert (A.contains(Key3)==false);

A.insert(Key3, 2.4);

assert (A.empty()==false);

assert (A.size()== 2);

assert (A.contains(Key2)==true);

assert (A.contains(Key3)==true);

//after deleting 1 element

A.erase(Key2);

assert (A.size()== 1);

assert (A.contains(Key2)==false);

assert (A.contains(Key3)==true);

A.update(Key3, 9.9);

assert (A.contains(Key3)==true);

assert (A.size()== 1);

assert (A.empty()==false);

cerr << "empty() --> passed" << endl;

cerr << "size() --> passed" << endl;

cerr << "contains() --> passed" << endl;

}

1. **Testing Erase Function**

void testErase (Map A)

{

// If key is equal to a key currently in the map, remove the key/value

// pair with that key from the map and return true. Otherwise, make

// no change to the map and return false.

KeyType Key1 = "one";

KeyType Key2 = "two";

KeyType Key3 = "three";

KeyType Key4 = "four";

KeyType Key5 = "five";

KeyType Key6 = "six";

//key is in the map, only entry

KeyType tempKeya = "default";

ValueType tempVala = 10.1;

bool a = A.insertOrUpdate(Key1, 1.2);

assert (a == true);

bool e1 = A.erase(Key1);

assert (e1 == true);

assert (A.size() == 0);

assert (A.contains(Key1) == false);

bool get1 = A.get(0, tempKeya, tempVala);

assert(!get1 && tempKeya == "default" && tempVala == 10.1);

a = A.insertOrUpdate(Key1, 1.2);

bool b = A.insertOrUpdate(Key2, 2.3);

bool c = A.insertOrUpdate(Key3, 3.4);

bool d = A.insertOrUpdate(Key4, 4.5);

bool e = A.insertOrUpdate(Key5, 5.6);

assert (b == true);

assert (a == true);

assert (c == true);

assert (d == true);

assert (e == true);

//key is not in map

bool e2 = A.erase(Key6);

assert(!e2);

assert (A.size() == 5);

assert (A.contains(Key6) == false);

for (int i = -1; i < A.size() +1; i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = A.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "one" && tempVal == 1.2);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == Key2 && tempVal == 2.3);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 3)

{

assert (x == true);

assert (tempKey == Key4 && tempVal == 4.5);

}

else if (i == 4)

{

assert (x == true);

assert (tempKey == Key5 && tempVal == 5.6);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 6)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

//key is in the map, first position

bool e3 = A.erase(Key1);

assert (e3);

assert (A.size() == 4);

assert (A.contains(Key1) == false);

for (int i = 0; i < A.size() +1; i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = A.get(i, tempKey, tempVal);

if (i == 0)

{

assert (x == true);

assert (tempKey == "two" && tempVal == 2.3);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == Key3 && tempVal == 3.4);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "four" && tempVal == 4.5);

}

else if (i == 3)

{

assert (x == true);

assert (tempKey == Key5 && tempVal == 5.6);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

//key is in the map, last entry

bool e4 = A.erase(Key5);

assert (e4);

assert (A.size() == 3);

assert (A.contains(Key5) == false);

for (int i = 0; i < A.size(); i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = A.get(i, tempKey, tempVal);

if (i == 0)

{

assert (x == true);

assert (tempKey == "two" && tempVal == 2.3);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == Key3 && tempVal == 3.4);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "four" && tempVal == 4.5);

}

else if (i == 3)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

//key is in the map, middle entry

bool e5 = A.erase(Key3);

assert (e5);

assert (A.size() == 2);

assert (A.contains(Key3) == false);

for (int i = 0; i < A.size(); i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = A.get(i, tempKey, tempVal);

if (i == 0)

{

assert (x == true);

assert (tempKey == "two" && tempVal == 2.3);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == "four" && tempVal == 4.5);

}

else if (i == 2)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

cerr << "Erase() --> passed" << endl;

}

1. **Testing swap Function**

void testSwap (Map A, Map B)

{

//one is empty, the other isn't empty

KeyType Key1 = "one";

KeyType Key2 = "two";

KeyType Key3 = "three";

KeyType Key4 = "four";

KeyType Key5 = "five";

KeyType Key6 = "six";

bool a = A.insertOrUpdate(Key1, 1.2);

bool b = A.insertOrUpdate(Key2, 2.3);

bool c = A.insertOrUpdate(Key3, 3.4);

bool d = A.insertOrUpdate(Key4, 4.5);

bool e = A.insertOrUpdate(Key5, 5.6);

assert (b == true);

assert (a == true);

assert (c == true);

assert (d == true);

assert (e == true);

assert (B.size() == 0);

A.swap(B);

//make sure that they have each other's values

assert (A.size() == 0);

assert (B.size() == 5);

KeyType tempKeyo = "default";

ValueType tempValo = 10.1;

bool al = A.get(0, tempKeyo, tempValo);

assert (!al);

assert (tempKeyo == "default" && tempValo == 10.1);

for (int i = -1; i < B.size() +1; i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = B.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "one" && tempVal == 1.2);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == Key2 && tempVal == 2.3);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 3)

{

assert (x == true);

assert (tempKey == Key4 && tempVal == 4.5);

}

else if (i == 4)

{

assert (x == true);

assert (tempKey == Key5 && tempVal == 5.6);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

//both are empty

Map C;

A.swap(C);

assert (C.size() == 0);

assert (A.size() == 0);

bool al1 = A.get(0, tempKeyo, tempValo);

assert (!al1);

assert (tempKeyo == "default" && tempValo == 10.1);

bool al2 = C.get(0, tempKeyo, tempValo);

assert (!al2);

assert (tempKeyo == "default" && tempValo == 10.1);

//they both the same

Map D;

D = B;

B.swap(D);

assert (D.size() == 5);

assert (B.size() == 5);

for (int i = -1; i < B.size() +1; i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = B.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "one" && tempVal == 1.2);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == Key2 && tempVal == 2.3);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 3)

{

assert (x == true);

assert (tempKey == Key4 && tempVal == 4.5);

}

else if (i == 4)

{

assert (x == true);

assert (tempKey == Key5 && tempVal == 5.6);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

for (int i = -1; i < D.size() +1; i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = D.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "one" && tempVal == 1.2);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == Key2 && tempVal == 2.3);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 3)

{

assert (x == true);

assert (tempKey == Key4 && tempVal == 4.5);

}

else if (i == 4)

{

assert (x == true);

assert (tempKey == Key5 && tempVal == 5.6);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

//they both have things

Map F;

bool ax = F.insertOrUpdate(Key1, 1.2);

bool bx = F.insertOrUpdate(Key2, 2.3);

bool cx = F.insertOrUpdate(Key3, 3.4);

assert (bx == true);

assert (ax == true);

assert (cx == true);

F.swap(D);

assert (D.size() == 3);

assert (F.size() == 5);

for (int i = -1; i < F.size() +1; i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = F.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "one" && tempVal == 1.2);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == Key2 && tempVal == 2.3);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 3)

{

assert (x == true);

assert (tempKey == Key4 && tempVal == 4.5);

}

else if (i == 4)

{

assert (x == true);

assert (tempKey == Key5 && tempVal == 5.6);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

for (int i = -1; i < D.size() +1; i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = D.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "one" && tempVal == 1.2);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == Key2 && tempVal == 2.3);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 3)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 4)

{

assert (x == true);

assert (tempKey == Key5 && tempVal == 5.6);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

cerr << "swap() --> passed" << endl;

}

1. **Test Subtract Function**

void testSubtractfunct(Map A, Map B)

{

//result isn't empty

Map result1;

assert (result1.empty() == true);

assert (result1.size() ==0);

//result is empty

//test if A and B are empty

Map result2;

KeyType Key1r = "un";

KeyType Key2r = "duo";

KeyType Key3r = "twa";

KeyType Key4r = "cat";

bool a = result2.insertOrUpdate(Key1r, 1.0);

bool b = result2.insertOrUpdate(Key2r, 2.0);

bool c = result2.insertOrUpdate(Key3r, 3.0);

bool d = result2.insertOrUpdate(Key4r, 4.0);

assert (a == true);

assert (b == true);

assert (c == true);

assert (d == true);

Subtract\_BothEmptyParam(A, B, result1, result2);

/\*assert (result1.empty() == true);

assert (result1.size() ==0);

assert (result2.empty() == true);

assert (result2.size() ==0);\*/

//key is only in A

//key is in both -> same value

//key is in both -> different valuess

KeyType Key1 = "one";

KeyType Key2 = "two";

KeyType Key3 = "three";

KeyType Key4 = "four";

KeyType Key5 = "five";

KeyType Key6 = "six";

KeyType Key7 = "seven";

bool ax = A.insertOrUpdate(Key1, 1.2); //unique to A

bool cx = A.insertOrUpdate(Key3, 3.4); //both same values

bool dx = A.insertOrUpdate(Key5, 4.5); //both maps, but different values

bool bx = A.insertOrUpdate(Key2, 2.3); //unique to A

bool gx = A.insertOrUpdate(Key6, 6.7); //unique to A

assert (ax == true && bx == true && cx == true && dx == true && gx == true);

assert (A.size() == 5 );

assert (A.empty() == false);

//key is only in B

//key is in both -> same value

//key is in both -> different valuess

bool ex = B.insertOrUpdate(Key5, 5.6); //in both maps, but different value

bool x1 = B.insertOrUpdate(Key7, 7.8);//unique to B

bool cx2 = B.insertOrUpdate(Key3, 3.4); //both same values

bool dx2 = B.insertOrUpdate(Key4, 99.9); //unique to B

assert (ex == true && cx2 == true && dx2 == true && x1 ==true);

assert (B.size() == 4 && B.empty() == false);

Subtract\_Normal(A, B, result1, result2);

Subtract\_SameValue(A, B, result1, result2);

Subtract\_SameObject(A, B, result1, result2);

Subtract\_OneEmptyParam (A, B, result1, result2);

cerr << "subtract() --> passed" << endl;

}

void Subtract\_OneEmptyParam (Map A, Map B, Map result1, Map result2)

{

Map C;

subtract(C, B, result1);

subtract(B, C, result2);

assert (result1.size() == 0 && result1.empty() == true);

assert (result2.size() == 4 && result2.empty() == false);

//test result 1

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = result1.get(0, tempKey, tempVal);

assert (x == false);

assert (tempKey == "default" && tempVal== 10.1);

//test result2

for (int i = -1; i < result2.size() +1; i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = result2.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "five" && tempVal == 5.6);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == "seven" && tempVal == 7.8);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 3)

{

assert (x == true);

assert (tempKey == "four" && tempVal == 99.9);

}

else if (i == 4)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

}

//set B equal to As

void Subtract\_SameValue(Map A, Map B, Map result1, Map result2)

{

A = B;

subtract(A, B, result1);

subtract(B, A, result2);

assert (result1.size() == 0 && result1.empty() == true);

assert (result2.size() == 0&& result2.empty() == true);

KeyType tempKey = "default";

KeyType tempKey2 = "default2";

ValueType tempVal = 10.1;

ValueType tempVal2 = 10.12;

bool x = result1.get(0, tempKey, tempVal);

assert (x == false);

assert (tempKey == "default" && tempVal== 10.1);

bool y = result2.get(0, tempKey2, tempVal2);

assert (y == false);

assert (tempKey2 == "default2" && tempVal2 == 10.12);

}

//set B equal to A

void Subtract\_SameObject(Map A, Map B, Map result1, Map result2)

{

subtract(A, A, result1);

subtract(B, B, result2);

assert (result1.size() == 0&& result1.empty() == true);

assert (result2.size() == 0 && result2.empty() == true);

KeyType tempKey = "default";

KeyType tempKey2 = "default2";

ValueType tempVal = 10.1;

ValueType tempVal2 = 10.12;

bool x = result1.get(0, tempKey, tempVal);

assert (x == false);

assert (tempKey == "default" && tempVal== 10.1);

bool y = result2.get(0, tempKey2, tempVal2);

assert (y == false);

assert (tempKey2 == "default2" && tempVal2 == 10.12);

}

//both have nothing

void Subtract\_BothEmptyParam (Map A, Map B, Map result1, Map result2)

{

subtract(A, B, result1);

subtract(B, A, result2);

assert (result1.size() == 0 && result1.empty() == true);

assert (result2.size() == 0 && result2.empty() == true);

KeyType tempKey = "default";

KeyType tempKey2 = "default2";

ValueType tempVal = 10.1;

ValueType tempVal2 = 10.12;

bool x = result1.get(0, tempKey, tempVal);

assert (x == false);

assert (tempKey == "default" && tempVal== 10.1);

bool y = result2.get(0, tempKey2, tempVal2);

assert (y == false);

assert (tempKey2 == "default2" && tempVal2 == 10.12);

}

void Subtract\_Normal (Map A, Map B, Map result1, Map result2)

{

subtract(A, B, result1);

subtract(B, A, result2);

assert(result1.size() == 3);

assert(result2.size() == 2);

for (int i = -1; i < result1.size(); i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = result1.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "one" && tempVal == 1.2);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == "two" && tempVal == 2.3);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "six" && tempVal == 6.7);

}

else if (i == 3)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

for (int i = 0; i < result2.size(); i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = result2.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "seven" && tempVal == 7.8);

}

if (i == 1)

{

assert (x == true);

assert (tempKey == "four" && tempVal == 99.9);

}

}

}

1. **Test Combine Function**

void testCombine (Map A, Map B)

{

//result isn't empty

Map result1;

assert (result1.empty() == true);

assert (result1.size() ==0);

//result is empty

//test if A and B are empty

combine(A, B, result1);

Map result2;

KeyType Key1r = "un";

KeyType Key2r = "duo";

KeyType Key3r = "twa";

KeyType Key4r = "cat";

bool a = result2.insertOrUpdate(Key1r, 1.0);

bool b = result2.insertOrUpdate(Key2r, 2.0);

bool c = result2.insertOrUpdate(Key3r, 3.0);

bool d = result2.insertOrUpdate(Key4r, 4.0);

assert (a == true);

assert (b == true);

assert (c == true);

assert (d == true);

combine(A, B, result1);

combine(A, B, result2);

assert (result1.empty() == true);

assert (result1.size() ==0);

assert (result2.empty() == true);

assert (result2.size() ==0);

//key is only in A

//key is in both -> same value

//key is in both -> different valuess

KeyType Key1 = "one";

KeyType Key2 = "two";

KeyType Key3 = "three";

KeyType Key4 = "four";

KeyType Key5 = "five";

bool ax = A.insertOrUpdate(Key1, 1.2); //unique to A

bool bx = A.insertOrUpdate(Key2, 2.3); //unique to A

bool cx = A.insertOrUpdate(Key3, 3.4); //both same values

bool dx = A.insertOrUpdate(Key4, 4.5); //both different values

assert (ax == true);

assert (bx == true);

assert (cx == true);

assert (dx == true);

assert (A.size() == 4 && A.empty() == false);

//key is only in B

//key is in both -> same value

//key is in both -> different valuess

bool ex = B.insertOrUpdate(Key5, 5.6); //unique to B

bool cx2 = B.insertOrUpdate(Key3, 3.4); //both same values

bool dx2 = B.insertOrUpdate(Key4, 99.9); //both different values

assert (ex == true);

assert (cx2 == true);

assert (dx2 == true);

assert (B.size() == 3 && B.empty() == false);

combineSameParam(A,B, result1, result2);

CombineNormal(A, B, result1, result2);

combineSameObject(A, B, result1, result2);

//combineSame(A, result1, result2);

cerr << "Combine() --> passed" << endl;

}

//SAME THING

void combineSameObject (Map A, Map B, Map result1, Map result2)

{

A = B;

bool combine1 = combine(B, B, result1);

bool combine2 = combine(A, A, result2);

assert (combine1 == true);

assert (combine2 == true);

assert (result1.size() == 3);

assert (result1.size() == 3);

/\*

bool ex = B.insertOrUpdate(Key5, 5.6); //unique to B

bool cx2 = B.insertOrUpdate(Key3, 3.4); //both same values

bool dx2 = B.insertOrUpdate(Key4, 99.9);\*/

for (int i = -1; i < result1.size(); i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = result1.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "five" && tempVal == 5.6);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "four" && tempVal == 99.9);

}

else if (i == 3)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 4)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 6)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

for (int i = 0; i < result2.size(); i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = result2.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "five" && tempVal == 5.6);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "four" && tempVal == 99.9);

}

else if (i == 3)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 4)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 6)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

}

//both param the same

void combineSameParam (Map A, Map B, Map result1, Map result2)

{

A = B;

bool combine1 = combine(A, B, result1);

bool combine2 = combine(B, A, result2);

assert (combine1 != false);

assert (combine2 != false);

assert (result1.size() == 3);

assert (result1.size() == 3);

/\*

bool ex = B.insertOrUpdate(Key5, 5.6); //unique to B

bool cx2 = B.insertOrUpdate(Key3, 3.4); //both same values

bool dx2 = B.insertOrUpdate(Key4, 99.9);\*/

for (int i = -1; i < result1.size(); i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = result1.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "five" && tempVal == 5.6);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "four" && tempVal == 99.9);

}

else if (i == 3)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 4)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 6)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

for (int i = 0; i < result2.size(); i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = result2.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "five" && tempVal == 5.6);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "four" && tempVal == 99.9);

}

else if (i == 3)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 4)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 6)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

}

//one param is empty

void combineOneEmptyParam (Map B, Map result1, Map result2)

{

Map C;

bool combine1 = combine(C, B, result1);

bool combine2 = combine(B, C, result2);

assert (combine1 == true && combine2 == true);

cerr << result1.size();

assert (result1.size() == 3);

assert (result1.size() == 3);

for (int i = -1; i < result1.size(); i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = result1.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "five" && tempVal == 5.6);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "four" && tempVal == 99.9);

}

else if (i == 3)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 4)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 6)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

for (int i = 0; i < result2.size(); i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = result2.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "five" && tempVal == 5.6);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "four" && tempVal == 99.9);

}

else if (i == 3)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 4)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 6)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

}

//both param have things

void CombineNormal (Map A, Map B, Map result1, Map result2)

{

bool combine1 = combine(A, B, result1);

bool combine2 = combine(A, B, result2);

assert (combine1 == false && combine2 == false);

assert (result1.size() == 4);

assert (result1.size() == 4);

for (int i = 0; i < result1.size(); i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = result1.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "one" && tempVal == 1.2);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == "two" && tempVal == 2.3);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 3)

{

assert (x == true);

assert (tempKey == "five" && tempVal == 5.6);

}

else if (i == 4)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 6)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

for (int i = 0; i < result2.size(); i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = result2.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "one" && tempVal == 1.2);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == "two" && tempVal == 2.3);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 3)

{

assert (x == true);

assert (tempKey == "five" && tempVal == 5.6);

}

else if (i == 4)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 6)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

}

1. **Test Insert Function**

void testInsert (Map A)

{

KeyType Key1 = "a";

KeyType Key2 = "b";

KeyType Key3 = "a";

//insert new key, new value -> true

//insert new key, same value -> false

//insert already key, new value -> false

//insert empty key, any value -> true

assert (A.size()==0);

assert (A.empty()==true);

//insert new key, new value -> true

bool x = A.insert (Key1, 1.2);

assert (A.size()==1);

assert (A.empty()==false);

assert (x == true);

assert (A.insertOrUpdate(Key1, 1.2) == true);

assert (A.contains(Key1));

//insert new key, same value -> false

ValueType val;

bool y = A.insert (Key1, 1.2);

assert (A.size()==1);

assert (A.empty()==false);

assert (y == false);

assert (A.insertOrUpdate (Key1, 1.2) == true);

assert (A.get(Key1, val) == true && val == 1.2);

//insert already key, new value -> false

ValueType valz;

bool yz = A.insert (Key1, 99.9);

assert (A.size()==1);

assert (A.empty()==false);

assert (yz == false);

assert (A.get(Key1, valz) == true && valz == 1.2);

//insert already key, new value -> false

ValueType val2 = 1.0;

bool z = A.insert (Key3, 1.3);

assert (A.size()==1);

assert (A.empty()==false);

assert (z == false);

assert (A.get(Key1, val2) == true && val2 == 1.2);

//normal, new key w/ new value

ValueType val3 = 10.1;

bool za = A.insert (Key2, 1.5);

assert (A.size()==2);

assert (A.empty()==false);

assert (za == true);

assert (A.insertOrUpdate (Key2, 1.5) == true);

assert (A.get(Key2, val3) == true && val3 == 1.5);

cerr << "Insert() --> passed" << endl;

}

1. **Test Update Function**

void testUpdate (Map A)

{

KeyType Key1 = "a";

KeyType Key2 = "b";

KeyType Key3 = "a";

ValueType val = 0;

//updating an empty map, will not work

bool x = A.update(Key1, 1.2);

assert (A.size()==0);

assert (A.empty()==true);

assert (x == false);

assert (A.contains(Key1) == false);

assert (A.get(Key1, val) == false && Key1 == "a" && val == 0);

//add an item

bool z = A.insert(Key1,1.2);

assert (A.size()==1);

assert (A.empty()==false);

assert (z == true);

assert (A.contains(Key1) == true);

assert (A.get(Key1, val) == true && Key1 == "a" && val == 1.2);

//update it, exiting key w/ new value

bool y = A.update (Key1, 1.3);

assert (A.size()==1);

assert (A.empty()==false);

assert (y == true);

assert (A.contains(Key1) == true);

assert (A.get(Key1, val) == true && Key1 == "a" && val == 1.3);

//update it, exiting key w/ existing value

bool yz = A.update (Key1, 1.3);

assert (A.size()==1);

assert (A.empty()==false);

assert (yz == true);

assert (A.contains(Key1) == true);

assert (A.get(Key1, val) == true && Key1 == "a" && val == 1.3);

cerr << "Update() --> passed" << endl;

}

1. **Test InsertorUpdate Function**

void testInsertOrUpdate (Map A)

{

// If key is equal to a key currently in the map, then make that key no

// longer map to the value it currently maps to, but instead map to

// the value of the second parameter; return true in this case.

// Otherwise, make no change to the map and return false.

KeyType Key1 = "a";

KeyType Key2 = "b";

KeyType Key3 = "a";

//UPDATE FUNCTIONS

ValueType val = 0;

//updating an empty map, will not work

bool x = A.insertOrUpdate(Key1, 1.2);

assert (A.size()==1);

assert (A.empty()==false);

assert (x == true);

assert (A.contains(Key1) == true);

assert (A.get(Key1, val) == true && Key1 == "a" && val == 1.2);

//update it, exiting key w/ new value

bool y = A.insertOrUpdate (Key1, 1.3);

assert (A.size()==1);

assert (A.empty()==false);

assert (y == true);

assert (A.contains(Key1) == true);

assert (A.get(Key1, val) == true && Key1 == "a" && val == 1.3);

//update it, exiting key w/ new value

bool yz = A.insertOrUpdate (Key1, 1.3);

assert (A.size()==1);

assert (A.empty()==false);

assert (yz == true);

assert (A.contains(Key1) == true);

assert (A.get(Key1, val) == true && Key1 == "a" && val == 1.3);

//INSERT FUNCTIONS

ValueType val1 = 0;

bool xinsert = A.insertOrUpdate (Key2, 1.5);

assert (A.size()==2);

assert (A.empty()==false);

assert (xinsert == true);

assert (A.contains(Key1) == true);

assert (A.get(Key2, val1) == true && Key2 == "b" && val1 == 1.5);

//basically just update

ValueType val2 = .998;

bool yza = A.insertOrUpdate (Key1, 1.009);

assert (A.size()==2);

assert (A.empty()==false);

assert (yza == true);

assert (A.get(Key1, val2) == true && val2 == 1.009);

//insert empty string as key, new value -> false

ValueType valzz;

KeyType EmptyrStr = "";

bool yzab = A.insertOrUpdate (EmptyrStr, 99.9);

assert (A.size()==3);

assert (A.empty()==false);

assert (yzab == true);

assert (A.get(EmptyrStr, valzz) == true && valzz == 99.9);

cerr << "InsertOrUpdate() --> passed" << endl;

}

1. **Test Get Functions**

void testGet2 (Map A)

{

// If 0 <= i < size(), copy into the key and value parameters the

// key and value of one of the key/value pairs in the map and return

// true. Otherwise, leave the key and value parameters unchanged and

// return false.

KeyType Key1 = "one";

KeyType Key2 = "two";

KeyType Key3 = "three";

KeyType Key4 = "four";

KeyType Key5 = "five";

bool a = A.insertOrUpdate(Key1, 1.2);

bool b = A.insertOrUpdate(Key2, 2.3);

bool c = A.insertOrUpdate(Key3, 3.4);

bool d = A.insertOrUpdate(Key4, 4.5);

bool e = A.insertOrUpdate(Key5, 5.6);

assert (a == true);

assert (b == true);

assert (c == true);

assert (d == true);

assert (e == true);

for (int i = -1; i < A.size() +1; i++)

{

KeyType tempKey = "default";

ValueType tempVal = 10.1;

bool x = A.get(i, tempKey, tempVal);

if (i == -1)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

if (i == 0)

{

assert (x == true);

assert (tempKey == "one" && tempVal == 1.2);

}

else if (i == 1)

{

assert (x == true);

assert (tempKey == Key2 && tempVal == 2.3);

}

else if (i == 2)

{

assert (x == true);

assert (tempKey == "three" && tempVal == 3.4);

}

else if (i == 3)

{

assert (x == true);

assert (tempKey == Key4 && tempVal == 4.5);

}

else if (i == 4)

{

assert (x == true);

assert (tempKey == Key5 && tempVal == 5.6);

}

else if (i == 5)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

else if (i == 6)

{

assert (x == false);

assert (tempKey == "default" && tempVal == 10.1);

}

}

cerr << "Get2() --> passed" << endl;

}

void testGet1(Map A)

{

// If key is equal to a key currently in the map, set value to the

// value in the map that that key maps to and return true. Otherwise,

// make no change to the value parameter of this function and return

// false.

assert (A.size()==0);

assert (A.empty()==true);

KeyType Key1 = "a";

KeyType Key2 = "b";

//key is in map, return true

ValueType val;

//updating an empty map, will not work

bool x = A.insertOrUpdate(Key1, 1.2);

assert (A.size()==1);

assert (A.empty()==false);

assert (x == true);

assert (A.contains(Key1) == true);

assert (A.get(Key1, val) == true && Key1 == "a" && val == 1.2);

//key isn't in map, return false

assert (A.get(Key2, val) == false && Key2 == "b" && val == 1.2);

ValueType val2 = 0;

assert (A.get(Key2, val2) == false && Key2 == "b" && val2 == 0);

cerr << "Get1() --> passed" << endl;

}