

```

/*Author: Eric Gustin
Assignment: CPSC223-01 HW07 hw7_tests.cpp
Description: This program tests the hash_table collection. It tests
every function in hash_table_collection, and accounts for many edge cases.
*/

```

```

#include <iostream>
#include <string>
#include <gtest/gtest.h>
#include "hash_table_collection.h"

```

```

using namespace std;

```

```

// Test 1
TEST(BasicsListTest, CorrectSize) {
    HashTableCollection<string,double> a;

    ASSERT_EQ(a.size(), 0);
    a.insert("AAPL", 220.29);
    ASSERT_EQ(a.size(), 1);
    a.insert("AMZN", 1739.81);
    ASSERT_EQ(a.size(), 2);
    a.insert("FB", 179.50);
    ASSERT_EQ(a.size(), 3);
    a.insert("GOOGL", 1240.54);
    ASSERT_EQ(a.size(), 4);
    a.insert("NFLX", 263.86);
    ASSERT_EQ(a.size(), 5);

    HashTableCollection<string,double> z;

    z = a;
    ASSERT_EQ(a.size(), z.size());

    // make sure doesn't drop to -1
    HashTableCollection<string,int> zero_list;
    zero_list.remove("nothing");
    ASSERT_EQ(zero_list.size(), 0);
}

```

```

// Test 2
TEST(BasicListTest, InsertAndFind) {
    HashTableCollection<string,double> b;
    b.insert("e", 50.0);
    b.insert("f", 60.0);
    b.insert("g", 70.0);
    b.insert("a", 10.0);
    b.insert("b", 20.0);
    b.insert("i", 90.0);
    b.insert("j", 100.0);
    b.insert("k", 110.0);
    b.insert("l", 120.0);
    b.insert("c", 30.0);
    b.insert("d", 40.0);
    b.insert("h", 80.0);
    b.insert("m", 130.0);
    b.insert("o", 140.0);

    double my_key1;
    ASSERT_EQ(b.find("a", my_key1), true);
    ASSERT_EQ(10.0, my_key1);
    double my_key2;
    ASSERT_EQ(b.find("o", my_key2), true);
    ASSERT_EQ(140.0, my_key2);
    double my_key3;
    ASSERT_EQ(b.find("f", my_key3), true);
    ASSERT_EQ(60.0, my_key3);
    double my_key4 = 0.0;
    ASSERT_EQ(b.find("z", my_key4), false);
    ASSERT_EQ(0.0, my_key4);
}

```

```

// Test 3
TEST(BasicListTest, RemoveElems) {
    HashTableCollection<string,double> c;
    // attempt to remove from an empty object
    c.remove("");
    c.insert("a", 10.0);
    c.insert("b", 20.0);
    c.insert("c", 30.0);
    c.insert("d", 40.0);
    c.insert("e", 50.0);
    c.insert("f", 60.0);

    ASSERT_EQ(c.size(), 6);
    c.remove("q"); // try to remove an element that is not in c. should do nothing
    ASSERT_EQ(c.size(), 6);
    c.remove("a"); // remove first element

    double v;
    ASSERT_EQ(c.find("a", v), false);
    ASSERT_EQ(c.find("b", v), true);
    ASSERT_EQ(v, 20.0);
    ASSERT_EQ(c.find("c", v), true);
    ASSERT_EQ(v, 30.0);
    ASSERT_EQ(c.find("d", v), true);
    ASSERT_EQ(v, 40.0);
    ASSERT_EQ(c.find("e", v), true);
    ASSERT_EQ(v, 50.0);
    ASSERT_EQ(c.find("f", v), true);
    ASSERT_EQ(v, 60.0);

    c.remove("f"); // remove last element

    ASSERT_EQ(c.size(), 4);
    ASSERT_EQ(c.find("f", v), false);
    c.remove("d"); // remove non edge element
    ASSERT_EQ(c.size(), 3);
    ASSERT_EQ(c.find("d", v), false);

    c.remove("b");

    c.remove("c");
    c.remove("e"); // remove the only element in the list
    ASSERT_EQ(c.size(), 0);
}

// Test 4
TEST(BasicListTest, GetKeys) {
    HashTableCollection<string,double>* d = new HashTableCollection<string,double>;
    d->insert("a", 10.0);
    d->insert("b", 20.0);
    d->insert("c", 30.0);
    vector<string> ks;
    d->keys(ks);
    vector<string>::iterator iter;
    iter = find(ks.begin(), ks.end(), "a");
    ASSERT_NE(iter, ks.end());
    iter = find(ks.begin(), ks.end(), "b");
    ASSERT_NE(iter, ks.end());
    iter = find(ks.begin(), ks.end(), "c");
    ASSERT_NE(iter, ks.end());
    iter = find(ks.begin(), ks.end(), "d");
    ASSERT_EQ(iter, ks.end());
    delete d;
}

// Test 5
TEST(BasicListTest, GetKeyRange) {
    HashTableCollection<string,double>* e = new HashTableCollection<string,double>;
    e->insert("a", 10.0);
    e->insert("b", 20.0);
    e->insert("c", 30.0);
    e->insert("d", 40.0);
    e->insert("e", 50.0);

```

```

vector<string> ks;

e->find("b", "d", ks);
vector<string>::iterator iter;
iter = find(ks.begin(), ks.end(), "b");
ASSERT_NE(iter, ks.end());
iter = find(ks.begin(), ks.end(), "c");
ASSERT_NE(iter, ks.end());
iter = find(ks.begin(), ks.end(), "d");
ASSERT_NE(iter, ks.end());
iter = find(ks.begin(), ks.end(), "a");
ASSERT_EQ(iter, ks.end());
iter = find(ks.begin(), ks.end(), "e");
ASSERT_EQ(iter, ks.end());
delete e;
// test if find range works for strings
// longer than one character
HashTableCollection<string,double>* f = new HashTableCollection<string,double>;
f->insert("apples", 10);
f->insert("golf", 20);
f->insert("hydro", 30);
f->insert("zebras", 40);

vector<string> ks2;
f->find("bees", "yoyo", ks2);
vector<string>::iterator iter2;
iter = find(ks2.begin(), ks2.end(), "golf");
ASSERT_NE(iter, ks2.end());
iter = find(ks2.begin(), ks2.end(), "hydro");
ASSERT_NE(iter, ks2.end());
iter = find(ks2.begin(), ks2.end(), "apples");
ASSERT_EQ(iter, ks2.end());
iter = find(ks2.begin(), ks2.end(), "zebras");
ASSERT_EQ(iter, ks2.end());
ASSERT_EQ(ks2.size(), 2);

// find only 1 key
vector<string> ks3;
f->find("zebras", "zebras", ks3);
ASSERT_EQ(ks3.size(), 1);
ASSERT_EQ(ks3[0], "zebras");
// find no keys
vector<string> ks4;
f->find("bye", "apple", ks4);
ASSERT_EQ(ks4.size(), 0);
delete f;
}

// Test 6
TEST(BasicListTest, KeySort) {
HashTableCollection<string,double>* g = new HashTableCollection<string,double>;
g->insert("e", 50.0);
g->insert("a", 10.0);
g->insert("d", 40.0);
g->insert("b", 20.0);
g->insert("c", 30.0);

// check to see if it is already sorted (it should be)
vector<string> pre_sorted_ks;
for(int i = 0; i < int(pre_sorted_ks.size()) - 1; ++i)
    ASSERT_LE(pre_sorted_ks[i], pre_sorted_ks[i+1]);

vector<string> sorted_ks;
g->sort(sorted_ks);
// check if sort order
for(int i = 0; i < int(sorted_ks.size()) - 1; ++i)
    ASSERT_LE(sorted_ks[i], sorted_ks[i+1]);
delete g;

HashTableCollection<string,double>* h = new HashTableCollection<string,double>;
h->insert("anagrams", 23);
h->insert("hemmingson", 1);
h->insert("regis", 99);

```

```

h->insert("salem oregon", 98);
h->insert("string", 44);
h->insert("turing", 23);
vector<string> sorted_stringInt;
h->sort(sorted_stringInt);
//check if sort order
for (int i = 0; i < int(sorted_stringInt.size())-1; ++i)
    ASSERT_LE(sorted_stringInt[i], sorted_stringInt[i+1]);
delete h;
}

TEST(BasicListTest, CopyList) {
    HashTableCollection<string,int> w;
    w.insert("goodbye", 11);
    w.insert("computer", 12);
    w.insert("forever", 122);
    w.insert("never", 2110);
    HashTableCollection<string,int> w_copy = w;
    ASSERT_EQ(w_copy.size(), w.size());

    vector<string> w_keys;
    vector<string> w_copy_keys;
    w.sort(w_keys);
    w_copy.sort(w_copy_keys);
    for (int i = 0; i <= 3; ++i)
    {
        ASSERT_EQ(w_keys[i], w_copy_keys[i]);
    }
    // copy constructor with empty hash table
    HashTableCollection<string,int> empty;
    HashTableCollection<string,int> empty_copy(empty);
    ASSERT_EQ(empty.size(), 0);
    ASSERT_EQ(empty_copy.size(), 0);
}

TEST(BasicListTest, Assign) {
    HashTableCollection<string,int> y;
    y.insert("g", 144);
    y.insert("ss", 344);
    y.insert("xp", 533);
    y.insert("er", 332);
    y.insert("a", 340);
    y.insert("go", 530);

    HashTableCollection<string,int> z;
    z.insert("China", 5631);
    z.insert("Non", 6335);
    z.insert("TV", 3320);
    z = y;
    vector<string> z_keys;
    vector<string> y_keys;
    y.sort(y_keys);
    z.sort(z_keys);
    ASSERT_EQ(z.size(), 6);
    for (int i = 0; i <= 5; ++i)
    {
        ASSERT_EQ(y_keys[i], z_keys[i]);
    }
    // assignment operator will not do copying if trying to copy itself.
    z = z;
}

TEST(BasicListTest, Negatives) {
    HashTableCollection<double,string>* l = new HashTableCollection<double,string>;
    l->insert(999.0, "DigitalLogic");
    l->insert(400.4, "AlgsAndDataStruct");
    l->insert(-33.2, "discreteMath");
    l->insert(-0.1, "Globals");
    l->insert(0.0, "Human Nature");

    vector<double> sorted_ints;
    l->sort(sorted_ints);
    for (int i = 0; i < int(sorted_ints.size())-1; ++i)

```

```

    {
        ASSERT_LE(sorted_ints[i], sorted_ints[i+1]);
    }
    delete l;
}

TEST(BasicListTest, SizeZero) {
    HashTableCollection<int,int>* m = new HashTableCollection<int,int>;
    ASSERT_EQ(m->size(), 0);
    m->remove(2);
    ASSERT_EQ(m->size(), 0);
    int my_val;
    ASSERT_EQ(m->find(0, my_val), false);
    vector<int> keys_ints;
    m->keys(keys_ints);
    ASSERT_EQ(keys_ints.size(), 0);
    m->sort(keys_ints);
    ASSERT_EQ(keys_ints.size(), 0);
    delete m;
}

TEST(BasicListTest, ResizeRehash) {
    HashTableCollection<int,int> n;
    // test to see if the hashtable rehashes
    for (int i = 0; i < 256; ++i) {
        n.insert(i, i+500);
    }
    ASSERT_EQ(n.size(), 256);

    vector<int> my_keys;
    n.keys(my_keys);

    int j = 255;
    while (n.size() > 0) {
        n.remove(my_keys[j]);
        --j;
    }
    ASSERT_EQ(n.size(), 0);
    vector<int> my_keys2;
    n.keys(my_keys2);
    ASSERT_EQ(my_keys2.size(), 0);
}

TEST(BasicListTest, BigDataSet) {
    HashTableCollection<int,int> o;
    // insert 50,000 KV pairs
    for (int i = 0; i < 50000; ++i)
        o.insert(i, i+400);
    vector<int> my_keys3;
    o.keys(my_keys3);
    ASSERT_EQ(my_keys3.size(), 50000);
    // remove 50,000 KV pairs
    int j = 50000;
    while (j >= 0) {
        o.remove(my_keys3[j]);
        --j;
    }
    ASSERT_EQ(o.size(), 0);
}

int main(int argc, char** argv)
{
    testing::InitGoogleTest(&argc, argv);
    return RUN_ALL_TESTS();
}

```