**Map.cpp**

#include "map.h"

#include <string>

#include <iostream>

#include <assert.h>

using namespace std;

Map::Map(){

cout<<"Default constructor is called"<<endl;

head = nullptr;

length =0;

}

bool Map::empty() const{

return (length ==0);

}

//input: none

//output: returns true if the Map is empty; else false

//side effects: none

void Map:: operator=(const Map& mapped){

cout<<"Equal operator called"<<endl;

node\* a = head;

while(a!=nullptr){

node \*b = a->next;

delete a;

a = b;

length--;

}

head = nullptr;

length = 0;

node\* c = mapped.head;

while(c!= nullptr){

add(c->data);

c = c->next;

}

}

int Map::size() const{

return length;

}

//input: none

//output: returns the number of elements in the Map

//side effects: none

bool Map::add(ElementType element){

if(!find(element.first)){

node\* newNode = new node;

newNode->data = element;

if (length==0){

newNode->next = head;

head = newNode;

}else{

node\* p = head;

// cout<<"This is point j"<<endl;

while((p->next!= nullptr)&&(p->next->data.first<p->data.first)){

p = p->next;

}

// cout<<"This is point k"<<endl;

newNode->next = p->next;

// cout<<"This is point l"<<endl;

p->next = newNode;

}

// cout<<"This is point m"<<endl;

length++;

return true;

}else{

return false;

}

}

//input: an element (key and its associated value)

//output: returns true if the key-value pair was

// added; else false(the key was found)

//side effects: Map has one more element (if addition

// done)

bool Map::find(KeyType key) const{

if (length!=0){

node\* p = head;

while(p!=nullptr){

if (p->data.first == key){

return true;

}else{

p = p->next;

}

}

}

return false;

}

ValueType Map::retrieve(KeyType key) const{

assert(find(key));

if (length!=0){

node\* p = head;

while(p!=nullptr){

if (p->data.first == key){

return p->data.second;

}else{

p = p->next;

}

}

}

return NULL;

}

//input: a key

//output: returns the value associated with key (a

// null pointer if the key was not found)

//side effects: none

bool Map::remove(KeyType key){

if (find(key)){

node\* p = head;

node\* a = nullptr;

while(p!=nullptr){

if (p->data.first == key){

a->next= p->next;

delete p;

length--;

return true;

}else{

a = p;

p = p->next;

}

}

}

return false;

}

Map::Map(const Map & mapped){

cout<<"Map copy constructor is called"<<endl;

head = nullptr;

length = 0;

node\* a = mapped.head;

while(a!= nullptr){

add(a->data);

a = a->next;

}

}

Map::~Map(){

cout<<"Map destructor is called"<<endl;

node\* a = head;

while(a!=nullptr){

node \*b = a->next;

delete a;

a = b;

length--;

}

}

void Map::displayAll() const{

node \*p = head;

int item = 1;

if (length==0){

cout<<"Map is empty"<<endl;

}

while(p!= nullptr){

string\* str = p->data.second;

cout<<"List item "<<item<<" key: "<<p->data.first;

cout<<" value: "<<str<<endl;

item++;

p = p->next;

}

}

//input: a key

//output: returns true if item is removed;

// else false (key not found)

//side effects: Map has one less element (if remove

// done)

//void map::displayAll() const;

// write contents of the Map to cout

**Map.cpp**

#ifndef Map\_

#define Map\_

#include "mapInterface.h"

class Map : public mapInterface {

public:

Map();

//Default constructor

~Map();

//Destructor

Map(const Map & mapped);

//Copy Constructor

bool empty() const;

//input: none

//output: returns true if the Map is empty; else false

//side effects: none

int size() const;

//input: none

//output: returns the number of elements in the Map

//side effects: none

bool add(ElementType element);

//input: an element (key and its associated value)

//output: returns true if the key-value pair was

// added; else false(the key was found)

//side effects: Map has one more element (if addition

// done)

bool find(KeyType key) const;

//input: a key

//output: return true is an element with this key was

// found; else false

//side effects: none

ValueType retrieve(KeyType key) const;

//input: a key

//output: returns the value associated with key (a

// null pointer if the key was not found)

//side effects: none

bool remove(KeyType key);

//input: a key

//output: returns true if item is removed;

// else false (key not found)

//side effects: Map has one less element (if remove

// done)

void operator=(const Map& mapped);

//Equal operator

void displayAll() const;

//input:none

//output: none

//side effects: Displays content of program

private:

class node{

public:

node \*next;

ElementType data;

};

int length;

node \*head;

};

#endif

**mapInterface.h**

//mapInterface.h

#ifndef MAP\_INTERFACE\_

#define MAP\_INTERFACE\_

/\*

A Map is a value oriented container class that holds a collection of elements related to each other by membership. Elements are key-value pairs.

\*/

#include<utility> //needed for the type: pair<T1, T2>

#include<string>

using namespace std;

typedef int KeyType;

//KeyType must be comparable using the

//relational operators

typedef string\* ValueType;

typedef pair<KeyType, ValueType> ElementType;

class mapInterface{

public:

virtual bool empty() const = 0;

//input: none

//output: returns true if the Map is empty; else false

//side effects: none

virtual int size() const = 0;

//input: none

//output: returns the number of elements in the Map

//side effects: none

virtual bool add(ElementType element) = 0;

//input: an element (key and its associated value)

//output: returns true if the key-value pair was

// added; else false(the key was found)

//side effects: Map has one more element (if addition

// done)

virtual bool find(KeyType key) const = 0;

//input: a key

//output: return true is an element with this key was

// found; else false

//side effects: none

virtual ValueType retrieve(KeyType key) const = 0;

//input: a key

//output: returns the value associated with key (a

// null pointer if the key was not found)

//side effects: none

virtual bool remove(KeyType key) = 0;

//input: a key

//output: returns true if item is removed;

// else false (key not found)

//side effects: Map has one less element (if remove

// done)

};

#endif

**MapTester.cpp**

#include "map.h"

#include <string>

#include <iostream>

#include <assert.h>

using namespace std;

bool TestConstructor();

//Used to Test copy constructor and default constructor

bool TestOperator(Map& map);

//Test Equal operator

bool TestAdd(Map& map, string\*& temp);

//Tests add function

bool TestSameKey(Map& map);

//Tests to see if program rejects a pair with same key

bool TestEmpty(Map& map);

//Tests if program is empty

int TestSize(Map& map);

//Checks size of the program

bool TestFind(Map& map);

//Tests find method of map

bool TestRemove(Map& map);

//Tests remove method of the map

void MethodOrganizer();

//Runs all the different tests

bool TestRetrieve(Map& map,string\*& temp);

//Tests the retrieve method of the map

int main(){

MethodOrganizer();

}

bool TestConstructor(){

Map map;

string str6 = "Copy";

pair<int, string\*> p6 = {7, &str6};

map.add(p6);

cout<<"Key in map being copied: 7 Value: "<<&str6<<endl;

Map newmap(map);

newmap.displayAll();

return true;

}

bool TestOperator(Map& map){

Map newmap;

map = newmap;

return (map.size() ==0);

}

bool TestAdd(Map& map, string\*& temp){

string str1 = "Hope";

string str2 = "that";

string str3 = "this";

string str4 = "works";

temp = &str1;

pair<int, string\*> p1 = {0, &str1};

pair<int, string\*> p2 = {5, &str2};

pair<int, string\*> p3 = {2, &str3};

pair<int, string\*> p4 = {3, &str4};

return (map.add(p1)&&map.add(p2)&&map.add(p3)&&map.add(p4));

}

bool TestSameKey(Map& map){

string str5 = "Shouldnt add";

pair<int, string\*> p5 = {2, &str5};

return !map.add(p5);

}

bool TestEmpty(Map& map){

return map.empty();

}

int TestSize(Map& map){

return map.size();

}

bool TestFind(Map& map){

return map.find(3);

}

bool TestRemove(Map& map){

return map.remove(3);

}

bool TestRetrieve(Map& map,string\*& temp){

return (temp == map.retrieve(0));

}

void MethodOrganizer(){

string\* temp;

cout<<"Testing copy constructor"<<endl;

TestConstructor();

Map map;

if (TestEmpty(map)){

cout<<"The empty functions works on an empty map"<<endl;

if (TestSize(map)==0){

cout<<"The size functions works on an empty map"<<endl;

cout<<"Displaying items before adding"<<endl;

map.displayAll();

if (TestAdd(map,temp)){

cout<<"Adding three items"<<endl;

map.displayAll();

if(TestSameKey(map)){

cout<<"Rejected adding item with same key"<<endl;

if(TestRetrieve(map,temp)){

cout<<"Retrieve works"<<endl;

}else{

cout<<"Temp:"<<temp<<" retrieve:"<<map.retrieve(0)<<endl;

}

if(TestFind(map)){

cout<<"Find function works on item in list"<<endl;

map.displayAll();

if(TestRemove(map)){

cout<<"Removing key 3"<<endl;

map.displayAll();

if(TestSize(map)==3){

cout<<"Size function works on nonempty list"<<endl;

if(TestEmpty(map)==false){

cout<<"Empty funciton works on nonempty list"<<endl;

map.displayAll();

cout<<"Removing key 3 which is not in map"<<endl;

if(TestRemove(map)==false){

map.displayAll();

if(TestFind(map)==false){

cout<<"Find function returned false for a key not in list"<<endl;

if(TestOperator(map)){

cout<<"Replacing map with new map"<<endl;

map.displayAll();

cout<<"Testing destructor"<<endl;

}else{

cout<<"Check operator"<<endl;

}

}else{

cout<<"Check find fucnction"<<endl;

}

}else{

cout<<"Check Remove"<<endl;

}

}else{

cout<<"Empty has failed"<<endl;

}

}else{

cout<<"Second Size has failed"<<map.size()<<endl;

}

}else{

cout<<"Remove failed"<<endl;

}

}else{

cout<<"Find has failed"<<endl;

}

}else{

cout<<"Test SameKey has failed"<<endl;

}

}else{

cout<<"Add has failed"<<endl;

}

}else{

cout<<"First size has falied"<<endl;

}

}else{

cout<<"Empty has falied"<<endl;

}

}

//bool TestRetrieve(Map& map);

**Makefile**

ass5: Map.o MapTester.o

g++ Map.o MapTester.o -o ass5

Map.o: map.cpp

g++ -c -std=c++11 map.cpp

MapTester.o: MapTester.cpp

g++ -c -std=c++11 MapTester.cpp

**Output**

Testing copy constructor

Default constructor is called

Key in map being copied: 7 Value: 0x7ffeeb3705a0

Map copy constructor is called

List item 1 key: 7 value: 0x7ffeeb3705a0

Map destructor is called

Map destructor is called

Default constructor is called

The empty functions works on an empty map

The size functions works on an empty map

Displaying items before adding

Map is empty

Adding three items

List item 1 key: 0 value: 0x7ffeeb370438

List item 2 key: 3 value: 0x7ffeeb3703e0

List item 3 key: 2 value: 0x7ffeeb3703f8

List item 4 key: 5 value: 0x7ffeeb370420

Rejected adding item with same key

Retrieve works

Find function works on item in list

List item 1 key: 0 value: 0x7ffeeb370438

List item 2 key: 3 value: 0x7ffeeb3703e0

List item 3 key: 2 value: 0x7ffeeb3703f8

List item 4 key: 5 value: 0x7ffeeb370420

Removing key 3

List item 1 key: 0 value: 0x7ffeeb370438

List item 2 key: 2 value: 0x7ffeeb3703f8

List item 3 key: 5 value: 0x7ffeeb370420

Size function works on nonempty list

Empty funciton works on nonempty list

List item 1 key: 0 value: 0x7ffeeb370438

List item 2 key: 2 value: 0x7ffeeb3703f8

List item 3 key: 5 value: 0x7ffeeb370420

Removing key 3 which is not in map

List item 1 key: 0 value: 0x7ffeeb370438

List item 2 key: 2 value: 0x7ffeeb3703f8

List item 3 key: 5 value: 0x7ffeeb370420

Find function returned false for a key not in list

Default constructor is called

Equal operator called

Map destructor is called

Replacing map with new map

Map is empty

Testing destructor

Map destructor is called