```
/**
 * ******* Lasalle College Vancouver *******.
 *
 * Object Oriented Programming in C++ II
 * Week 8 - STL Associative Containers: map, unordered map
 * @author
 * Ivaldo Tributino de Sousa <ISousa@lasallecollegevancouver.com>
 */
#pragma once
// Input/output library
#include <iostream>
using std :: cout;
using std :: endl;

// Containers library
#include<vector>
#include<forward_list>
#include<map>
#include<unordered_map>
using std :: vector;
using std :: forward_list;
using std :: map;
using std :: unordered_map;
using std::pair;

// Strings library
#include <string>
using std :: string;
using std :: to_string;

// Numerics library
#include <cmath>

// Utilities library
# include <utility>
# include <functional>
```

calculator.h

```
struct Calculator {

    // typedef allows the programmer to create new names for types
    // MFP f compile equally as double (*f)(int,int);
    typedef double (Calculator::*MFP)(int,int);

    map <char, MFP> fmap;

    double add(int a, int b) {return a+b;}
    double subtract(int a, int b) {return a-b;}
    double multiply(int a, int b) {return a*b;}
    double divide(int a, int b) {return (b == 0)? INFINITY : double(a)/b;}

    Calculator() {
        // &class_name::member_function_name
        fmap.insert( std::make_pair( '+', &Calculator::add ));
        fmap.insert( std::make_pair( '-', &Calculator::subtract));
        fmap.insert( std::make_pair( '*', &Calculator::multiply));
        fmap.insert( std::make_pair( '/', &Calculator::divide));
    }

    double Call( const char & c, int x, int y ) {
        MFP fp = fmap[c];
        return (this->*fp)(x,y);
    }
};
```

hashTable.h

```
class Pair {
public:
    int key;
    int value;
    Pair(int key, int v) : key(key), value(v){}

};

class HashTable {

private:

    forward_list<Pair>* t;

public:

    HashTable();

    int HashFunc(int key);

    void Insert(int key, int value);

    void Search_key(int key);

    void Remove(int key);

    void Print_Table();

    ~HashTable();
};
```

hashTable.cpp

```
const int row = 10;
const int column = 10;

HashTable :: HashTable() : t(new forward_list<Pair>[column]){}


int HashTable :: HashFunc(int key)
{
    return key % column;
}

void HashTable :: Insert(int key, int value)
{
    if(key < column*row){
        int h = HashFunc(key);

        for(Pair &p : t[h]){
            if(p.key == key){
                p.value = value;
                return;
            }
        }
        t[h].emplace_front(Pair(key,value));
    }
    else{
        cout << "supports only keys 0 to " << column*row-1 << endl;
    }
}

void HashTable :: Search_key(int key)
{
    int h = HashFunc(key);
    for(Pair &p : t[h]){
        if(p.key == key){
            cout<< "value :" << p.value << endl;
            return;
        }
    }
    cout << "Key not found" << endl;

}

void HashTable :: Remove(int key)
{

}

void HashTable :: Print_Table(){
    for(int i = 0; i<column; ++i){
        cout << i << " ---> ";
        for(const Pair& p : t[i]){
            cout << "{" << p.key << " : " << p.value << "} ";
        }
        cout << '\n';
    }
}

HashTable :: ~HashTable()
{
    delete[] t;
}
```

polygon.h

```
class Polygon {

  private: // Private members:

    // Data Members (underscore indicates a private member variable)
    unsigned int numberSides_;

  protected: // Protected mebers:
    string solidName;

  public:  // Public members:
      /**
      * Creates a triangle.
      */
    Polygon(); // Custom default constructor

    /**
      * Creates a numberSides sided Polygon.
      */
    Polygon(int numberSides);

    /**
    * Copy constructor: creates a new Polygon from another.
    * @param obj polygon to be copied.
    */
    Polygon(const Polygon & obj); // Custom Copy constructor

    ~Polygon(); // Destructor

    Polygon & operator=(const Polygon & obj);  // Custom assignment operator;

    /**
      * Function Call Operator () Overloading:
      */
    double operator()(float lenght) const;

    bool operator==(const Polygon & obj) const;

    bool operator<(const Polygon & obj) const;

    bool operator>(const Polygon & obj) const;

    /**
    * Return the polygon name by its number of sides.
    */
    string shapeName() const;

    /**
      * Gets and Sets
      */

    unsigned int getNumberSides() const;

    void setNumberSides(unsigned int n);

};

namespace std {
template<>
struct std::hash<Polygon>
    {
        std::size_t operator()(const Polygon& poly) const noexcept
        {
            return  std::hash<std::string>{}(poly.shapeName());
        }
    };
}
```

polygon.cpp

```
#include "Polygon.h"

// #define Allows the programmer to give a name to a constant value before
the program is compiled
#define PI 3.14159265

Polygon :: Polygon() : numberSides_(3){
  cout << "Default Constructor Invoked"  << endl;
}

Polygon :: Polygon(int numberSides){
  (numberSides > 2)? numberSides_ = numberSides : numberSides_ = 3;
  cout << "Constructor Invoked"  << endl;
}

Polygon :: Polygon(const Polygon & obj){
  numberSides_ = obj.numberSides_;
  cout << "Copy Constructor Invoked"  << endl;
}

Polygon :: ~Polygon(){
  cout << "Polygon was destructive"  << endl;
}

Polygon & Polygon :: operator=(const Polygon & obj){
  numberSides_ = obj.numberSides_;
  cout << "Assignment operator invoked"  << endl;
  return *this;
}

// function to overload the operators
double Polygon :: operator()(float length) const{
  double perimeter = numberSides_*length;
  double apothem = (length)/(2*tan(PI/numberSides_));
  return perimeter*apothem/2;

}

bool Polygon :: operator==(const Polygon & obj) const{
  return numberSides_ == obj.numberSides_;
}

bool Polygon :: operator <(const Polygon & obj) const{

  return numberSides_ < obj.numberSides_;
}

bool Polygon :: operator >(const Polygon & obj) const{

  return numberSides_ > obj.numberSides_;
}

string Polygon::shapeName() const {

  string arrayName[6] = {"triangle" , "square", "pentagon",
  "hexagon", "heptagon", "octagon"};

  string name = (numberSides_<9)? arrayName[numberSides_-3]:
to_string(numberSides_)+"_polygon";

  return name;
}

unsigned int Polygon ::getNumberSides() const {
  return numberSides_;
}

void Polygon :: setNumberSides(unsigned int n){
  numberSides_ = (n > 2)? n : 3;
}
```

main.cpp

```cpp
int main(){

    //************************************************
    //         ---- std::map ::insert -----
    //************************************************

    map<int, int> fibonacci;

    fibonacci[0]; // operator[]
    fibonacci.insert(map<int, int>::value_type(1, 1));  // value_type
    fibonacci.insert(std::pair<int, int>(2, 1));      // pair
    fibonacci.insert({4,3}); // { , }
    fibonacci.insert(std::make_pair(3, 2));  // make_pair

    for (auto& pair: fibonacci) {
        std::cout << "{" << pair.first << " : " << pair.second << "}\n";
    }
    cout << "- By [k,v]" << '\n';

    for (auto& [k,v]: fibonacci) {
        std::cout << "{" << k << " : " << v << "}\n";
    }

    //************************************************
    //         ---- Map & Iterators -----
    //************************************************

    map<int,int>::iterator itor;
    for(itor = fibonacci.begin(); itor != fibonacci.end(); ++itor){
        cout << "key is: " << itor->first << " : ";
        cout << "Value is: " << itor->second << endl;
    }

    {
        map<string, Polygon> mapPolys;
        {
            cout << "- operator[]" << endl;
            mapPolys["Triangle"];
        }
        {
            cout << "- value_type" << endl;
            mapPolys.insert(map<string, Polygon>::value_type("Square", 
Polygon(4)));
        }
        {
            cout << "- pair" << endl;
            mapPolys.insert(std::pair("Pentagon", Polygon(5)));
        }
        {
            cout << "- make_pair" << endl;
            mapPolys.insert(std::make_pair("Hexagon", Polygon(6)));
        }
        {
            cout << "- { , }(curly braces)" << endl;
            mapPolys.insert({"Heptagon",Polygon(7)});
        }
        {
            cout << "- operator[]" << endl;
            mapPolys["Octagon"].setNumberSides(8);
        }

        map<string,Polygon> :: iterator it;
        it = mapPolys.find("nonagon");

        if(it != mapPolys.end()){
            cout << it->second.shapeName() << endl;
        }
        else{
            cout << "Out of range" << endl;
        }
```

```
    }

    //************************************************
    //         ---- Creating a map of lambdas -----
    //************************************************
    {

        map<string, std::function<double(float)>> areas;

        for(int i=3; i<8; i++){
            Polygon p(i);

            areas[p.shapeName()] = [=](float x){return p(x);};
        }

        cout << areas["square"](10) << '\n';

        for (const auto& pair: areas) {
            cout << "------------------------- " << pair.first << " 
-------------------------" << endl;
            for(int i =1; i<7;++i){
                std::cout << pair.second(i) << " | ";
            }
            cout << '\n';
        }

    }


    //************************************************
    // ---- std::map of member function pointers -----
    //************************************************
    cout << "---- std::map of member function pointers -----" << endl;
    Calculator C;
    cout << C.Call( '+', 3, 6) << endl;
    cout << C.Call( '-', 3, 6) << endl;
    cout << C.Call( '*', 3, 6) << endl;
    cout << C.Call( '/', 3, 6) << endl;
    cout << C.Call( '/', 3, 0) << endl;

    //*************************************************
    // ---- Map (Binary Tree) -----
    //*************************************************
    cout << "---- Map (Binary Tree) -----" << endl;
    {
        // This unconventional implementation is only intended to show the
importance of the operator <
        // for the method find().

        Polygon triangle(3);
        Polygon square(4);
        Polygon pentagon(5);
        Polygon hexagon(6);
        Polygon heptagon(7);

        map<Polygon, double> polyAreas;

        polyAreas[triangle] = 10.0;
        polyAreas[square] = 5.0;
        polyAreas[pentagon] = 5.7;
        polyAreas[hexagon] = 14.67;
        polyAreas[heptagon] = 90.78;

        for(const auto& pair : polyAreas){
            cout << pair.first.shapeName() << endl;
        }

        cout << "--- Using find method ---" << '\n';
        const Polygon& poly = hexagon;

        if(polyAreas.find(poly) != polyAreas.end()){
```

```
        cout << poly.shapeName() <<endl;
    }

    cout << "--- Sorting a map ---" << '\n';
    for(auto& p : polyAreas){
        cout << p.first.shapeName() << endl;
    }
}

//************************************************
//            ---- Hash Table -----
//************************************************
cout << "---- Hash Table -----" << endl;

// HashMapTable table;


HashTable t;

for(int i=0; i<100; ++i){
    t.Insert(i, 2*(i%10));
}

t.Remove(15);
t.Print_Table();



//************************************************
//      ---- EXAMPLE: unordered_map -----
//************************************************
cout << "---- EXAMPLE: unordered_map -----" << endl;

// Construction by assigning Initializer_list
unordered_map<string, double> umap({{"PI", 3.1416},{"Root2", 1.414}});

// inserting values by using [] operator
```

```
umap["root3"] = 1.732;
umap["log10"] = 2.302;

// inserting value by insert function
umap.insert(std::make_pair("e", 2.718));

// inserting value by emplace function
umap.emplace("loge",1.0);

// unordered_map Element Access
cout << umap["root3"] << endl;
// unordered_map Capacity
cout << umap.size() << endl;

// iterator find(const key_type& k)
unordered_map<string, double> :: iterator umapItor = umap.find("PI");

if(umapItor != umap.end())
{
    pair<string, double> pr = *umapItor;
    cout << pr.first << ", " << pr.second << endl;
}
// unordered_map bucket()

for(const auto& pr : umap){
    cout << pr.first << ", " << pr.second << endl;
    cout << "bucket number: " << umap.bucket(pr.first) << endl;
}

{
unordered_map<Polygon, double> unPolyAreas;

unPolyAreas[Polygon(3)] = 10.0;
unPolyAreas[Polygon(4)] = 5.0;
unPolyAreas[Polygon(5)] = 5.7;
unPolyAreas[Polygon(6)] = 14.67;
```

```
    unPolyAreas[Polygon(7)] = 90.78;

    for(auto& [k,v] : unPolyAreas){
        cout << "Area of a " << k.shapeName();
        cout << " with side lengths " << v;
        cout << ": " << k(v) << endl;
    }

    }
    return 0;
}
```