**Map:**

Maps are associative containers that store elements formed by combination of key value and mapped value.

Note:

Each key is unique and cannot be changed, it can be inserted or deleted but cannot be altered. Value associated with the key can be altered.

Key are arranged in ascending order.

Time complexiy: O(n)

**Creating a map**:

syntax:

*map<key\_type, value\_type> map\_name;*

***To insert:***

*Based on insert function:*

*//*map\_name.insert( std::pair< key\_type, value\_type>(key, value))*;*

*mymap.insert(std::pair<char, int> (‘a’, 10));*

*Based on iterator:*

*//Lets say ‘it’ is the iterator pointing to the begingin of mymap*

std::map<char,int>::iterator it = mymap.begin();

mymap.insert (it, std::pair<char,int>('b',300));

mymap.insert (it, std::pair<char,int>('c',400));

*Based on range:*

*std::map<char, int> anothermap;*

*anothermap.insert( mymap.begin(), mymap.end());*

***lower\_bound/ upper\_bound:***

lower\_bound, It retun to iterator where it is pointing to.

Eg,

it = myMap.lower\_bound(‘a’)

//When print is been made based on ‘a’ then we need to start from

upper\_bound, It return to iterator the next location to the key mentioned.

Eg,

it = myMap.upper\_bound(‘d’)

//But the i will point to ‘e’ but it will not print ‘e’ when print is made based on the //upper\_bound *.*

Note:

Key and value are inserted as pair.

map containers are slower when compared to unordered\_map containers

pair is a class template to store a pair of values and indvidual values can be accessed ny the keyword first and second.

**To erase**:

Based on key:

map\_name.erase(‘key’);

Based on iterator:

//say ‘it’ is iterator

it = map\_name.begin();

map\_name.erase(it);

Based on range:

it = map\_name.find(‘b’);

map\_name.erase(it, map\_name.end());

**Program**:

Pgm1: setting value and key based on ‘at’ keyword

#include<iostream>

#include<string>

#include<map>

int main()

{

//Create a map

std::map<std::string, int> myMap = {

{"alpha", 0},

{"beta", 0},

{"gama", 0}

};

//Accessig map based on key and setting value

myMap.at("alpha") = 10;

myMap.at("beta") = 20;

myMap.at("gama") = 30;

//Printing the value

for(auto&x: myMap){

std::cout << x.first << ": " << x.second << '\n';

}

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pgm2: How key and value are inserted and printed

#include <iostream>

#include <iterator>

#include <map>

int main()

{

// empty map container

std::map<int, int> map1;

// insert elements in random order

map1.insert(std::pair<int, int>(4, 40));

map1.insert(std::pair<int, int>(3, 30));

map1.insert(std::pair<int, int>(7, 70));

map1.insert(std::pair<int, int>(2, 20));

map1.insert(std::pair<int, int>(6, 60));

map1.insert(std::pair<int, int>(5, 50));

map1.insert(std::pair<int, int>(1, 10));

// printing map map1

std::map<int, int>::iterator itr;

std::cout << "\nThe map map1 is : \n";

std::cout << "\tKEY\tELEMENT\n";

for (itr = map1.begin(); itr != map1.end(); ++itr) {

std::cout << '\t' << itr->first

<< '\t' << itr->second << '\n';

}

std::cout << std::endl;

// assigning the elements from map1 to map2

std::map<int, int> map2(map1.begin(), map1.end());

// print all elements of the map map2

std::cout << "\nThe map map2 after"

<< " assign from map1 is : \n";

std::cout << "\tKEY\tELEMENT\n";

for (itr = map2.begin(); itr != map2.end(); ++itr) {

std::cout << '\t' << itr->first

<< '\t' << itr->second << '\n';

}

std::cout << std::endl;

// remove all elements up to

// element with key=3 in map2

std::cout << "\nmap2 after removal of"

" elements less than key=3 : \n";

std::cout << "\tKEY\tELEMENT\n";

// erase(0, n) will earse elements from index 0 to n-1

map2.erase(map2.begin(), map2.find(3));

for (itr = map2.begin(); itr != map2.end(); ++itr) {

std::cout << '\t' << itr->first

<< '\t' << itr->second << '\n';

}

// To erase element based on key

// remove all elements with key = 4

int num;

num = map2.erase(4);

std::cout << "\nmap2.erase(4) : ";

std::cout << num << " removed \n";

std::cout << "\tKEY\tELEMENT\n";

for (itr = map2.begin(); itr != map2.end(); ++itr) {

std::cout << '\t' << itr->first

<< '\t' << itr->second << '\n';

}

std::cout << std::endl;

// lower bound and upper bound for map map1 key = 5

std::cout << "map1.lower\_bound(5) : "

<< "\tKEY = ";

std::cout << map1.lower\_bound(5)->first << '\t';

std::cout << "\tELEMENT = "

<< map1.lower\_bound(5)->second << std::endl;

std::cout << "map1.upper\_bound(5) : "

<< "\tKEY = ";

std::cout << map1.upper\_bound(5)->first << '\t';

std::cout << "\tELEMENT = "

<< map1.upper\_bound(5)->second << std::endl;

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pgm3: Seting value based on key and finding the size

#include<iostream>

#include<map>

#include<string>

int main()

{

//Creating a map container with char as key and string as its value

std::map<char, std::string> myMap;

//map[key] = value;

myMap['a'] = "A element";

myMap['b'] = "Another element B";

myMap['c'] = myMap['b'];

//map[key] will print the value

std::cout << "map key 'a' has " << myMap['a'] << std::endl;

std::cout << "map key 'b' has " << myMap['b'] << std::endl;

std::cout << "map key 'c' has " << myMap['c'] << std::endl;

//when a map key is not present it does not print anything

std::cout << "map key 'd' has " << myMap['d'] << std::endl;

//map.size() will print the total size

std::cout << "myMap now contains " << myMap.size() << " elemetns\n";

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Pgm4: Insert in different mechanism

#include<iostream>

#include<map>

int main()

{

std::map<char, int> myMap;

//first insert function

myMap.insert(std::pair<char, int>('a', 100));

myMap.insert(std::pair<char, int>('z', 500));

//second insert functioin

std::map<char, int>::iterator it = myMap.begin();

myMap.insert(it, std::pair<char, int>('b', 300));

myMap.insert(it, std::pair<char, int>('c', 400));

//third inst function version

std::map<char, int> anothermap;

anothermap.insert(myMap.begin(), myMap.find('c'));

//Printing the contents

std::cout << "mymap contains:\n";

for(it = myMap.begin(); it != myMap.end(); it++){

std::cout << it->first << " => " << it->second << "\n";

}

std::cout << "anothermap contains:\n";

for(it = anothermap.begin(); it != anothermap.end(); it++)

std::cout << it->first << " => " << it->second << "\n";

return 0;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_