**Map**

**What?**

It’s a associative container that store element in key value pair.

**Why?**

1. It takes only unique elements that too in a sorted order based on assigned sorting criteria.

2. As key are in sorted order searching through map is faster.

3. There must be a value attached to a key.

4. It’s implemented using binary search tree.

Note:

When searching a element with the iterator, the iterator will return the element if found else it will return the end position mentioning that its not found.

**Insert:**

map.insert(data)

map[key] = value

**Search element in map by key:**

map.find(key) != map.end()

cout << found

map.find(key) == map.end()

cout << not found

**Erase element from map:**

map.erase(key)

**External comparator:**

In the data type we have to mention the class/struct name in which we overload the () operator which is used for sorting the map.

Eg,

struct **WordGreaterComparator**

{

bool operator()(const std::string & left, const std::string & right) const

{

return (left > right);

}

};

std::map<std::string, int, **WordGreaterComparator** > mapOfWords\_2;

mapOfWords\_2.insert(std::make\_pair("earth", 1));

mapOfWords\_2.insert(std::make\_pair("moon", 2));

mapOfWords\_2.insert(std::make\_pair("sun", 3));

for(std::map<std::string,int>::iterator it = mapOfWords\_2.begin(); it != mapOfWords\_2.end(); it++)

std::cout<<it->first<<" :: "<<it->second<<std::endl;

Note:

by default the map uses < operator for sorting key value.

For user defined tyoe the class needs the following for sorting,

1. Default sorting criteria i.e. operator < defined

2. std::map should be assigned with an external sorting criteria

Eg,

*class User*

*{*

*std::string m\_id;*

*std::string m\_name;*

*public:*

*User(std::string name, std::string id)*

*:m\_id(id), m\_name(name)*

*{}*

*const std::string& getId() const {*

*return m\_id;*

*}*

*const std::string& getName() const {*

*return m\_name;*

*}*

*bool operator< (const User& userObj) const*

*{*

*if(userObj.m\_id < this->m\_id)*

*return true;*

*}*

*};*

*void example\_1()*

*{*

*std::map<User, int> m\_UserInfoMap;*

*m\_UserInfoMap.insert(std::make\_pair<User, int>(User("Mr.X", "3"), 100) );*

*m\_UserInfoMap.insert(std::make\_pair<User, int>(User("Mr.X", "1"), 120) );*

*m\_UserInfoMap.insert(std::make\_pair<User, int>(User("Mr.Z", "2"), 300) );*

*std::map<User, int>::iterator it = m\_UserInfoMap.begin();*

*for(; it != m\_UserInfoMap.end(); it++)*

*{*

*std::cout<<it->first.getName()<<" :: "<<it->second<<std::endl;*

*}*

*}*

*Output :: Comparing by ID*

*Mr.X :: 100  
Mr.Z :: 300  
Mr.X :: 120*

**Iteration in map:**

std::map<type>:: iterator it;

it = mapName.begin();

while(it != mapName.end()){

it->first(); // To access the key

it->second(); // To access the value

it++; // Increment the iterator

}

**How to check is a key is present in map?**

Using count method

If(Map.count(key) > 0 )

cout << “present” ;

Here,

count() method is used to check whether a key is present or not.

Using find method

std::map<type>::iterator it;

it = map.find(key);

if(it != map.end()){

cout << “Key is present”;

}