In this article we see how & why to use std::map in c++.

std::map Introduction

std::map is an associative container that store elements in key-value pair.

Benefits of using std::map :

- It stores only unique keys and that too in sorted order based on its assigned sorting criteria.
- As keys are in sorted order therefore searching element in map through key is very fast i.e. it takes logarithmic time.
- In std::map there will be only one value attached with the every key.
- std::map can be used as associative arrays.
- It might be implemented using balanced binary trees.

Lets see an example,

```cpp
#include <iostream>
#include <map>
#include <string>
#include <iterator>

int main()
{
    std::map<std::string, int> mapOfWords;
    // Inserting data in std::map
    mapOfWords.insert(std::make_pair("earth", 1));
    mapOfWords.insert(std::make_pair("moon", 2));
    mapOfWords["sun"] = 3;
    // Will replace the value of already added key i.e. earth
    mapOfWords["earth"] = 4;
    // Iterate through all elements in std::map
    std::map<std::string, int>::iterator it = mapOfWords.begin();
    while(it != mapOfWords.end())
    {
        std::cout<<it->first<<" :: "<<it->second<<std::endl;
        it++;
    }
    // Check if insertion is successful or not
    if(mapOfWords.insert(std::make_pair("earth", 1)).second == fal
    {
        std::cout<<"Element with key 'earth' not inserted because
    }
    // Searching element in std::map by key.
    if(mapOfWords.find("sun") != mapOfWords.end())
        std::cout<<"word 'sun' found"<<std::endl;
    if(mapOfWords.find("mars") == mapOfWords.end())
        std::cout<<"word 'mars' not found"<<std::endl;
    return 0;
}
```

Output:

earth :: 4
moon :: 2
sun :: 3
Element with key ‘earth’ not inserted because already existed
word ‘sun’ found
word ‘mars’ not found

Creating std::map objects

Creating a std::map of words i.e.

Key = Word (std::string)
Value = Word’s frequency count (int)

```
std::map<std::string, int> mapOfWords;
```

As no external sorting criteria for key(std::string) is specified in above std::map, therefore it will use default key sorting criteria i.e operator < and all elements will be arranged inside std::map in alphabetical sorted order of keys.

Inserting data in std::map :

Inserting data using insert member function,

```
mapOfWords.insert(std::make_pair("earth", 1));
mapOfWords.insert(std::make_pair("moon", 2));
```

We can also insert data in std::map using operator [] i.e.

```
mapOfWords["sun"] = 3;
```

Different between operator [] and insert function:

If specified key already existed in map then operator [] will silently change its value where as insert will not replace already added key instead it returns the information i.e. if element is added or not. e.g.

```
mapOfWords["earth"] = 4; // Will replace the value of already adde
```

Where as for insert member function,

```
mapOfWords.insert(std::make_pair("earth", 1)).second
```

will return false.

Iterating through all std::map elements:

```
std::map<std::string, int>::iterator it = mapOfWords.begin();
while(it != mapOfWords.end())
{
std::cout<<it->first<<" :: "<<it->second<<std::endl;
it++;
}
```

Each entry in std::map<std::string, int> is std::pair<std::string, int> therefore through iterator, key can be accessed by it->first and value by it->second .

Searching element in std::map by key

find member function of std::map can be used to search element in std::map by key. If specified key is not present then it returns the std::map::end else an iterator to the searched element.

```
iterator find (const key_type& k);

//e.g.

if(mapOfWords.find("sun") != mapOfWords.end())
std::cout<<"word 'sun' found"<<std::endl;
if(mapOfWords.find("mars") == mapOfWords.end())
std::cout<<"word 'mars' not found"<<std::endl;
```

Searching element in std::map by Value

To search element in std::map by value we need to iterate through all of the elements and check for the passed value and return i.e.

```
#include <iostream>
#include <map>
#include <string>
#include <iterator>

std::map<std::string, int>::iterator serachByValue(std::map<std::s
{
    // Iterate through all elements in std::map and search for the
    std::map<std::string, int>::iterator it = mapOfWords.begin();
    while(it != mapOfWords.end())
    {
        if(it->second == val)
        return it;
        it++;
    }
}
int main()
{
    std::map<std::string, int> mapOfWords;
    // Inserting data in std::map
    mapOfWords.insert(std::make_pair("earth", 1));
    mapOfWords.insert(std::make_pair("moon", 2));
    mapOfWords["sun"] = 3;

    std::map<std::string, int>::iterator it = serachByValue(mapOfW
    if(it != mapOfWords.end())
        std::cout<<it->first<<" :: "<<it->second<<std::endl;

return 0;
}
```

Output:

sun :: 3

Deleting data from std::map

std::map’s erase member function is used to delete the element in std::map i.e.

```
void erase (iterator position);
size_type erase (const key_type& k);
void erase (iterator first, iterator last);
```

Code example,

```cpp
#include <map>
#include <string>
#include <iterator>
int main()
{
    std::map<std::string, int> mapOfWords;
    mapOfWords.insert(std::make_pair("earth", 1));
    mapOfWords.insert(std::make_pair("moon", 2));
    mapOfWords["sun"] = 3;

    // Erasing By iterator
    std::map<std::string, int>::iterator it = mapOfWords.find("moo
    mapOfWords.erase(it);

    // Erasing By Key
    mapOfWords.erase("earth");

    return 0;
}
```