

ThisPointer.com

[C++11 Tutorials](#)[STL](#)[C++ Tutorials](#)[Design Patterns](#)[Multithreading](#)[java](#)[GDB](#)[Datastructure ▾](#)[Home](#) » [C++](#) » [std::map](#) » std::map Tutorial Part 1: Usage Detail with examples

std::map Tutorial Part 1: Usage Detail with examples

👤 Varun 🕒 January 31, 2015 📄 C++, std::map 💬 1 Comment

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.

[Vector](#)[List](#)[Deque](#)[Set](#)[Map](#)[MultiMap](#)[STL Algorithms](#)

1.) What's std::vector and why to use it?

2.) Different ways to initialize a vector

3.) How does std::vector works internally

4.) User Defined Objects & std::vector

5.) How to use vector efficiently in C++?

6.) std::vector and Iterator Invalidation

7.) Remove repeated elements from a vector

8.) Fill a vector with random numbers

9.) Hidden cost of std::vector

10.) Adding elements in Vector

Lets see an example,

```
#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 == false)
    {
        std::cout<<"Element with key 'earth' not inserted because al
    }
    // 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

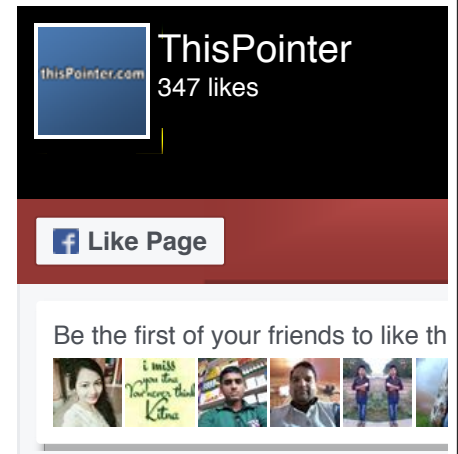
Creating std::map objects

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

Advertisements

Advertisements

Find Us On Facebook



Search

Search

Advertisements

Advertisements

Key = Word (std::string)

Value = Word's frequency count (int)

```
1 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,

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

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

```
1 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.

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

Where as for insert member function,

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

will return false.

Iterating through all std::map elements:

```

1 std::map<std::string, int>::iterator it = mapOfWords.begin();
2 while(it != mapOfWords.end())
3 {
4     std::cout<<it->first<<" :: "<<it->second<<std::endl;
5     it++;
6 }

```

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.

```

1 iterator find (const key_type& k);
2
3 //e.g.
4
5 if(mapOfWords.find("sun") != mapOfWords.end())
6     std::cout<<"word 'sun' found"<<std::endl;
7 if(mapOfWords.find("mars") == mapOfWords.end())
8     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.

```

1 #include <iostream>
2 #include <map>
3 #include <string>
4 #include <iterator>
5
6 std::map<std::string, int>::iterator serachByValue(std::map<std
7 {
8     // Iterate through all elements in std::map and search for th
9     std::map<std::string, int>::iterator it = mapOfWords.begin()
10    while(it != mapOfWords.end())
11    {
12        if(it->second == val)
13            return it;
14        it++;
15    }
16 }

```

```

17 int main()
18 {
19     std::map<std::string, int> mapOfWords;
20     // Inserting data in std::map
21     mapOfWords.insert(std::make_pair("earth", 1));
22     mapOfWords.insert(std::make_pair("moon", 2));
23     mapOfWords["sun"] = 3;
24
25     std::map<std::string, int>::iterator it = serachByValue(mapOfWords);
26     if(it != mapOfWords.end())
27         std::cout<<it->first<<" :: "<<it->second<<std::endl;
28
29     return 0;
30 }

```

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.

```

1 void erase (iterator position);
2 size_type erase (const key_type& k);
3 void erase (iterator first, iterator last);

```

Code example,

```

1 #include <iostream>
2 #include <map>
3 #include <string>
4 #include <iterator>
5 int main()
6 {
7     std::map<std::string, int> mapOfWords;
8     mapOfWords.insert(std::make_pair("earth", 1));
9     mapOfWords.insert(std::make_pair("moon", 2));
10    mapOfWords["sun"] = 3;
11
12    // Erasing By iterator
13    std::map<std::string, int>::iterator it = mapOfWords.find("earth");
14    mapOfWords.erase(it);
15
16    // Erasing By Key
17    mapOfWords.erase("earth");
18
19    return 0;
20 }

```

Related Posts:

- [std::map Tutorial -Part 2: std::map and External Sorting...](#)
- [C++ Set example and Tutorial – Part 1](#)
- [Set vs Map : How to choose a right associative container ?](#)
- [Different ways to insert elements in an unordered_map](#)
- [How to Iterate over a map in C++](#)
- [C++ Map Insert Example](#)
- [std::map Tutorial – Part 3: Using User defined class...](#)
- [C++ std::list Tutorial, Example and Usage Details](#)
- [How to find an element in unordered_map](#)
- [How to Erase / Remove an element from an unordered_map](#)
- [How to Insert elements in an unordered_set in C++11](#)
- [Unordered_map Usage Tutorial and Example](#)
- [multimap Example and Tutorial in C++](#)
- [How to iterate a map in reverse order – C++](#)
- [c++11 unordered_map : erase elements while iterating in a...](#)
- [c++ std::vector and Iterator Invalidation example](#)
- [How to erase elements from a list in c++ using iterators](#)
- [How to remove elements from a List while Iterating](#)
- [C++11 'auto' Tutorial and Examples](#)
- [How to Sort a Map by Value in C++](#)
- [How to iterate over a HashSet in Java](#)
- [Using unordered_set with custom hasher and comparision...](#)

- [std::unordered_set Basic Usage & Example](#)
- [c++ std::set example and tutorial with external Sorting...](#)
- [How to iterate over an unordered_map in C++11](#)

» C++, std::map, STL

1 Comment Already



[Subscribe to comments feed](#)



Warren - May 3rd, 2016 at 7:20 am

Thanks for the clear demonstration.

[Reply](#)

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

[« Create a Binary Search Tree from an array](#)

[std::map Tutorial -Part 2: std::map and External Sorting Criteria / Comparator »](#)

C++11 unordered_map

Basic Usage Detail and Example
Initializing an unordered_map
Searching in unordered_map
Insert elements in unordered_map
Erasing an element
Erase elements while iterating

Terms and Conditions

Terms and Conditions
Policy

C++11 Multithreading

Part 1: Three Ways to Create Threads
Part 2: Joining and Detaching Threads
Part 3: Passing Arguments to Threads
Part 4 : Sharing Data & Race Conditions
Part 5 : Fixing Race Conditions using mutex
Part 6 : Need of Event Handling
Part 7: Condition Variables
Part 8: std::future and std::promise

C++11 Smart Pointers

Part 1: shared_ptr Usage Details
Part 2: shared_ptr and Custom Deletor
Part 3: shared_ptr vs raw pointer
Part 4: Create shared_ptr objects carefully
Part 5: shared_ptr and Cyclic References

STL Interview Questions

std::sort Tutorial & Example
std::unique Tutorial & Example
1.) Using std::find & std::find_if with User Defined Classes
2.) Iterating over a range of User Defined objects and calling member function using std::for_each