Map in C++ Standard Template Library (STL)

Maps are associative containers that store elements in a mapped fashion. Each element has a key value and a mapped value. No two mapped values can have same key values.  
  
   
Some basic functions associated with Map:  
[begin()](https://www.geeksforgeeks.org/mapbegin-end-c-stl/) – Returns an iterator to the first element in the map  
[end()](https://www.geeksforgeeks.org/mapbegin-end-c-stl/) – Returns an iterator to the theoretical element that follows last element in the map  
[size()](https://www.geeksforgeeks.org/mapsize-c-stl/) – Returns the number of elements in the map  
[max\_size()](https://www.geeksforgeeks.org/map-max_size-in-c-stl/) – Returns the maximum number of elements that the map can hold  
[empty()](https://www.geeksforgeeks.org/mapempty-c-stl/) – Returns whether the map is empty  
[pair insert(keyvalue, mapvalue)](https://www.geeksforgeeks.org/map-insert-in-c-stl/) – Adds a new element to the map  
[erase(iterator position)](https://www.geeksforgeeks.org/map-erase-function-in-c-stl/) – Removes the element at the position pointed by the iterator  
[erase(const g)](https://www.geeksforgeeks.org/map-erase-function-in-c-stl/)– Removes the key value ‘g’ from the map  
[clear()](https://www.geeksforgeeks.org/mapclear-c-stl/) – Removes all the elements from the map

|  |
| --- |
| #include <iostream>  #include <iterator>  #include <map>    using namespace std;    int main()  {        // empty map container      map<int, int> gquiz1;        // insert elements in random order      gquiz1.insert(pair<int, int>(1, 40));      gquiz1.insert(pair<int, int>(2, 30));      gquiz1.insert(pair<int, int>(3, 60));      gquiz1.insert(pair<int, int>(4, 20));      gquiz1.insert(pair<int, int>(5, 50));      gquiz1.insert(pair<int, int>(6, 50));      gquiz1.insert(pair<int, int>(7, 10));        // printing map gquiz1      map<int, int>::iterator itr;      cout << "\nThe map gquiz1 is : \n";      cout << "\tKEY\tELEMENT\n";      for (itr = gquiz1.begin(); itr != gquiz1.end(); ++itr) {          cout << '\t' << itr->first               << '\t' << itr->second << '\n';      }      cout << endl;        // assigning the elements from gquiz1 to gquiz2      map<int, int> gquiz2(gquiz1.begin(), gquiz1.end());        // print all elements of the map gquiz2      cout << "\nThe map gquiz2 after"           << " assign from gquiz1 is : \n";      cout << "\tKEY\tELEMENT\n";      for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {          cout << '\t' << itr->first               << '\t' << itr->second << '\n';      }      cout << endl;        // remove all elements up to      // element with key=3 in gquiz2      cout << "\ngquiz2 after removal of"              " elements less than key=3 : \n";      cout << "\tKEY\tELEMENT\n";      gquiz2.erase(gquiz2.begin(), gquiz2.find(3));      for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {          cout << '\t' << itr->first               << '\t' << itr->second << '\n';      }        // remove all elements with key = 4      int num;      num = gquiz2.erase(4);      cout << "\ngquiz2.erase(4) : ";      cout << num << " removed \n";      cout << "\tKEY\tELEMENT\n";      for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {          cout << '\t' << itr->first               << '\t' << itr->second << '\n';      }        cout << endl;        // lower bound and upper bound for map gquiz1 key = 5      cout << "gquiz1.lower\_bound(5) : "           << "\tKEY = ";      cout << gquiz1.lower\_bound(5)->first << '\t';      cout << "\tELEMENT = "           << gquiz1.lower\_bound(5)->second << endl;      cout << "gquiz1.upper\_bound(5) : "           << "\tKEY = ";      cout << gquiz1.upper\_bound(5)->first << '\t';      cout << "\tELEMENT = "           << gquiz1.upper\_bound(5)->second << endl;        return 0;  } |

**Output:**

The map gquiz1 is :

KEY ELEMENT

1 40

2 30

3 60

4 20

5 50

6 50

7 10

The map gquiz2 after assign from gquiz1 is :

KEY ELEMENT

1 40

2 30

3 60

4 20

5 50

6 50

7 10

gquiz2 after removal of elements less than key=3 :

KEY ELEMENT

3 60

4 20

5 50

6 50

7 10

gquiz2.erase(4) : 1 removed

KEY ELEMENT

3 60

5 50

6 50

7 10

gquiz1.lower\_bound(5) : KEY = 5 ELEMENT = 50

gquiz1.upper\_bound(5) : KEY = 6 ELEMENT = 50

**List of all functions of Map:**

* [map insert() in C++ STL](https://www.geeksforgeeks.org/map-insert-in-c-stl/)– Insert elements with a particular key in the map container. .
* [map count() function in C++ STL](https://www.geeksforgeeks.org/map-count-function-in-c-stl/)– Returns the number of matches to element with key value ‘g’ in the map.
* [map equal\_range() in C++ STL](https://www.geeksforgeeks.org/map-equal_range-in-c-stl/)– Returns an iterator of pairs. The pair refers to the bounds of a range that includes all the elements in the container which have a key equivalent to k.
* [map erase() function in C++ STL](https://www.geeksforgeeks.org/map-erase-function-in-c-stl/)– Used to erase element from the container.
* [map rend() function in C++ STL](https://www.geeksforgeeks.org/map-rend-function-in-c-stl/)– Returns a reverse iterator pointing to the theoretical element right before the first key-value pair in the map(which is considered its reverse end).
* [map rbegin() function in C++ STL](https://www.geeksforgeeks.org/map-rbegin-function-in-c-stl-2/)– Returns a reverse iterator which points to the last element of the map.
* [map find() function in C++ STL](https://www.geeksforgeeks.org/map-find-function-in-c-stl/)– Returns an iterator to the element with key value ‘g’ in the map if found, else returns the iterator to end.
* [map crbegin() and crend() function in C++ STL](https://www.geeksforgeeks.org/map-crbegin-and-crend-function-in-c-stl/)– **crbegin()** returns a constant reverse iterator referring to the last element in the map container. **crend()** returns a constant reverse iterator pointing to the theoretical element before the first element in the map.
* [map cbegin() and cend() function in C++ STL](https://www.geeksforgeeks.org/map-cbegin-and-cend-function-in-c-stl/)– **cbegin()** returns a constant iterator referring to the first element in the map container. **cend()** returns a constant iterator pointing to the theoretical element that follows last element in the multimap.
* [map emplace() in C++ STL](https://www.geeksforgeeks.org/map-emplace-in-c-stl/)– Inserts the key and its element in the map container.
* [map max\_size() in C++ STL](https://www.geeksforgeeks.org/map-max_size-in-c-stl/)– Returns the maximum number of elements a map container can hold.
* [map upper\_bound() function in C++ STL](https://www.geeksforgeeks.org/map-upper_bound-function-in-c-stl/)– Returns an iterator to the first element that is equivalent to mapped value with key value ‘g’ or definitely will go after the element with key value ‘g’ in the map
* [map operator= in C++ STL](https://www.geeksforgeeks.org/map-operator-in-c-stl/)– Assigns contents of a container to a different container, replacing its current content.
* [map lower\_bound() function in C++ STL](https://www.geeksforgeeks.org/map-lower_bound-function-in-c-stl/)– Returns an iterator to the first element that is equivalent to mapped value with key value ‘g’ or definitely will not go before the element with key value ‘g’ in the map.
* [map emplace\_hint() function in C++ STL](https://www.geeksforgeeks.org/map-emplace_hint-function-in-c-stl/)– Inserts the key and its element in the map container with a given hint.
* [map value\_comp() in C++ STL](https://www.geeksforgeeks.org/map-value_comp-in-c-stl/)– Returns the object that determines how the elements in the map are ordered (‘<' by default).
* [map key\_comp() function in C++ STL](https://www.geeksforgeeks.org/map-key_comp-function-in-c-stl/)– Returns the object that determines how the elements in the map are ordered (‘<' by default).
* [map::size() in C++ STL](https://www.geeksforgeeks.org/mapsize-c-stl/)– Returns the number of elements in the map.
* [map::empty() in C++ STL](https://www.geeksforgeeks.org/mapempty-c-stl/)– Returns whether the map is empty.
* [map::begin() and end() in C++ STL](https://www.geeksforgeeks.org/mapbegin-end-c-stl/)– **begin()** returns an iterator to the first element in the map. **end()** returns an iterator to the theoretical element that follows last element in the map
* [map::operator[] in C++ STL](https://www.geeksforgeeks.org/map-operator-cpp-stl/)– This operator is used to reference the element present at position given inside the operator.
* [map::clear() in C++ STL](https://www.geeksforgeeks.org/mapclear-c-stl/)– Removes all the elements from the map.
* [map::at() and map::swap() in C++ STL](https://www.geeksforgeeks.org/mapat-mapswap-c-stl/)– **at()** function is used to return the reference to the element associated with the key k. **swap()** function is used to exchange the contents of two maps but the maps must be of same type, although sizes may differ.

**// REVERSE TRAVERSING OF MAP [ EXAMPLE ]**

#include<iostream>

#include<map>

#include<utility>

using namespace std ;

int main()

{

map<int, int> sampleMap;

sampleMap.insert(pair<int, int>(1, 40));

sampleMap.insert(pair<int, int>(5, 30));

sampleMap.insert(pair<int, int>(3, 60));

sampleMap.insert(pair<int, int>(2, 20));

sampleMap.insert(pair<int, int>(6, 50));

**map<int,int> :: iterator it ;**

cout<<"Normal traversing : "<<endl;

for(it=sampleMap.**begin()**;it!=sampleMap.**end()**;it++)

{ cout<<it->first << " " <<it->second <<endl ; }

COUT<<"Reverse traversing : "<<endl ;

**map<int,int> :: reverse\_iterator itr ;**

for(itr=sampleMap.**rbegin()** ; itr!=sampleMap.**rend()** ; itr++)

{ cout<<itr->first << " " <<itr->second <<endl ; }

return 0;

}

OUTPUT :

**Normal traversing :**

**1 40**

**2 20**

**3 60**

**5 30**

**6 50**

**Reverse traversing :**

**6 50**

**5 30**

**3 60**

**2 20**

**1 40**

***/\* Difference between ordered map and unordered\_map \*/***

map vs unordered\_map in C++

Pre-requisite : [std::map](https://www.geeksforgeeks.org/map-associative-containers-the-c-standard-template-library-stl/), [std::unordered\_map](https://www.geeksforgeeks.org/unordered_map-in-stl-and-its-applications/)

When it comes to efficiency, there is a huge difference between maps and unordered maps.  
We must know the internal working of both to decide which one is to be used.

**Difference :**

| map | unordered\_map

---------------------------------------------------------

Ordering | increasing order | no ordering

| (by default) |

Implementation | Self balancing BST | Hash Table

| like [Red-Black Tree](https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/) |

search time | log(n) | O(1) -> Average

| | O(n) -> Worst Case

Insertion time | log(n) + Rebalance | Same as search

Deletion time | log(n) + Rebalance | Same as search

**Use std::map when**

* You need ordered data.
* You would have to print/access the data (in sorted order).
* You need predecessor/successor of elements.
* See [advantages of BST over Hash Tabl](https://www.geeksforgeeks.org/advantages-of-bst-over-hash-table/)e for more cases.

|  |
| --- |
| // CPP program to demonstrate use of std::map  #include <bits/stdc++.h>    int main()  {      // Ordered map      std::map<int, int> order;        // Mapping values to keys      order[5] = 10;      order[3] = 5;      order[20] = 100;      order[1] = 1;        // Iterating the map and printing ordered values      for (auto i = order.begin(); i != order.end(); i++) {          std::cout << i->first << " : " << i->second << '\n';      }  } |

Output :

1 : 1

3 : 5

5 : 10

20 : 100

**Use std::unordered\_map when**

* You need to keep count of some data (Example – strings) and no ordering is required.
* You need single element access i.e. no traversal.

|  |
| --- |
| // CPP program to demonstrate use of  **// std::unordered\_map**  #include <bits/stdc++.h>    int main()  {      // Unordered map      std::unordered\_map<int, int> order;        // Mapping values to keys      order[5] = 10;      order[3] = 5;      order[20] = 100;      order[1] = 1;        // Iterating the map and printing unordered values      for (auto i = order.begin(); i != order.end(); i++) {          std::cout << i->first << " : " << i->second << '\n';      }  } |

Output :

1 : 1

3 : 5

20 : 100

5 : 10