unordered_map in C++ STL

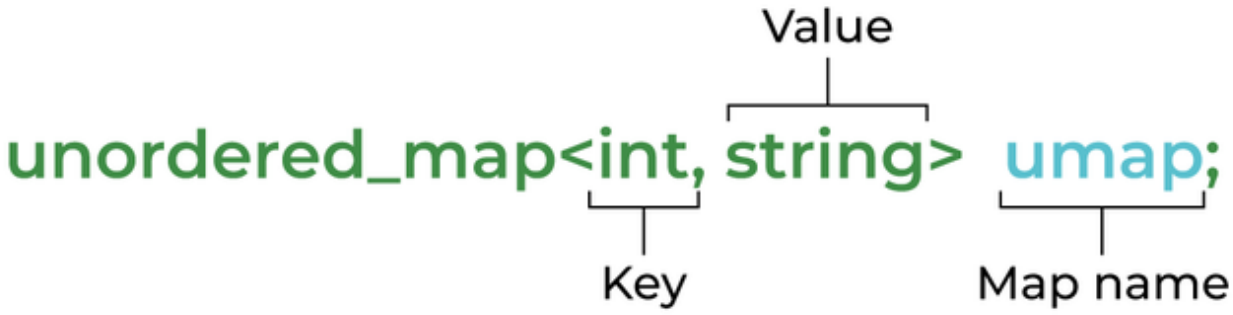
Difficulty Level : Easy • Last Updated : 10 Jan, 2023

unordered_map is an associated container that stores elements formed by the combination of a key value and a mapped value. The key value is used to uniquely identify the element and the mapped value is the content associated with the key. Both key and value can be of any type predefined or user-defined. In simple terms, an ***unordered_map*** is like a data structure of dictionary type that stores elements in itself. It contains successive pairs (key, value), which allows fast retrieval of an individual element based on its unique key.

Internally unordered_map is implemented using Hash Table, the key provided to map is hashed into indices of a hash table which is why the performance of data structure depends on the hash function a lot but on average, the cost of **search, insert, and delete** from the hash table is O(1).

> ***Note:*** *In the worst case, its time complexity can go from O(1) to O(n), especially for big prime numbers. In this situation, it is highly advisable to use a map instead to avoid getting a TLE(Time Limit Exceeded) error.*

Syntax:



Below is the C++ program to demonstrate an unordered map:


```
// C++ program to demonstrate
// functionality of unordered_map
#include <iostream>
#include <unordered_map>
using namespace std;

// Driver code
int main()
{
  // Declaring umap to be of
  // <string, int> type key
  // will be of STRING type
  // and mapped VALUE will
  // be of int type
  unordered_map<string, int> umap;

  // inserting values by using [] operator
  umap["GeeksforGeeks"] = 10;
  umap["Practice"] = 20;
  umap["Contribute"] = 30;

  // Traversing an unordered map
  for (auto x : umap)
    cout << x.first << " " <<
            x.second << endl;
}
```

Output

```
Contribute 30
Practice 20
GeeksforGeeks 10
```

Read Discuss Courses Practice Video

Key	Value
1	GFG
3	GFG_1
2	GFG_2
8	GFG_3
5	GFG_4

unordered_map Output

Explanation: The specific thing that this output justifies is that the value of the outcome of unordered_map is produced in a random key-to-value manner whereas the map displays value and key in an ordered manner.

unordered_map vs unordered_set

Unordered_map contains elements only in the form of (key-value) pairs.	Unordered_set does not necessarily contain elements in the form of key-value pairs, these are mainly used to see the presence/absence of a set.
Operator '**[]**' to extract the corresponding value of a key that is present in the map.	The searching for an element is done using a **find**() function. So no need for an operator[].

Note: For example, consider the problem of counting the frequencies of individual words. We can't use unordered_set (or set) as we can't store counts while we can use unordered_map.

unordered_map vs map

Unordered_map	Map
The unordered_map key can be stored in any order.	The map is an ordered sequence of unique keys
Unordered_Map implements an unbalanced tree structure due to which it is not possible to maintain order between the elements	Map implements a balanced tree structure which is why it is possible to maintain order between the elements (by specific tree traversal)
The time complexity of unordered_map operations is O(1) on average.	The time complexity of map operations is O(log n)

Methods on unordered_map

A lot of functions are available that work on unordered_map. The most useful of them

- **operator []**
- **empty**

- **begin and end for the iterator.**
- **find and count for lookup.**
- **insert and erase for modification.**

The below table shows the complete list of the methods of an unordered_map:

Methods/Functions	Description
at()	This function in C++ unordered_map returns the reference to the value with the element as key k
begin()	Returns an iterator pointing to the first element in the container in the unordered_map container
end()	Returns an iterator pointing to the position past the last element in the container in the unordered_map container
bucket()	Returns the bucket number where the element with the key k is located in the map
bucket_count	Bucket_count is used to count the total no. of buckets in the unordered_map. No parameter is required to pass into this function
bucket_size	Returns the number of elements in each bucket of the unordered_map
count()	Count the number of elements present in an unordered_map with a given key
equal_range	Return the bounds of a range that includes all the elements in the container with a key that compares equal to k

Methods/Functions	Description
find()	Returns iterator to the element
empty()	Checks whether the container is empty in the unordered_map container
erase()	Erase elements in the container in the unordered_map container

The C++11 library also provides functions to see internally used bucket count, bucket size, and also used hash function and various hash policies but they are less useful in real applications. We can iterate over all elements of unordered_map using Iterator.

C++

```
// C++ program to demonstrate
// Initialization, indexing,
// and iteration
#include <iostream>
#include <unordered_map>
using namespace std;

// Driver code
int main()
{
  // Declaring umap to be of
  // <string, double> type key
  // will be of string type and
  // mapped value will be of double type
  unordered_map<string, double> umap = {  //inserting element directly in map
  {"One", 1},
  {"Two", 2},
  {"Three", 3}
};

  // inserting values by using [] operator
  umap["PI"] = 3.14;
  umap["root2"] = 1.414;
  umap["root3"] = 1.732;
  umap["log10"] = 2.302;
  umap["loge"] = 1.0;

  // inserting value by insert function
```

```
  string key = "PI";

  Read    Discuss    Courses    Practice    Video
  if (umap.find(key) == umap.end())
    cout << key << " not found\n\n";

  // If key found then iterator to that
  // key is returned
  else
    cout << "Found " << key << "\n\n";

  key = "lambda";
  if (umap.find(key) == umap.end())
    cout << key << " not found\n";
  else
    cout << "Found " << key << endl;

  // iterating over all value of umap
  unordered_map<string, double>::iterator itr;
  cout << "\nAll Elements : \n";
  for (itr = umap.begin();
       itr != umap.end(); itr++)
  {
    // itr works as a pointer to
    // pair<string, double> type
    // itr->first stores the key part and
    // itr->second stores the value part
    cout << itr->first << "  " <<
            itr->second << endl;
  }
}
```

Output

```
Found PI


lambda not found


All Elements :
e  2.718
loge  1
log10  2.302
Two  2
One  1
Three  3
```

```
root2  1.414
root3  1.732
```


Find frequencies of individual words

Given a string of words, the task is to find the frequencies of the individual words:

Input: *str = "geeks for geeks geeks quiz practice qa for";*
Output: *Frequencies of individual words are*
(practice, 1)
(for, 2)
(qa, 1)
(quiz, 1)
(geeks, 3)

Below is the C++ program to implement the above approach:

C++

```
// C++ program to find freq
// of every word using unordered_map
#include <bits/stdc++.h>
using namespace std;

// Prints frequencies of
// individual words in str
void printFrequencies(const string &str)
{
  // declaring map of <string, int> type,
  // each word is mapped to its frequency
  unordered_map<string, int> wordFreq;

  // breaking input into word using
  // string stream
  // Used for breaking words
  stringstream ss(str);

  // To store individual words
  string word;
  while (ss >> word)
```

```
  // and printing them in <, > format
  unordered_map<string, int>:: iterator p;
```



```
                  p->second << ")\n";
}

// Driver code
int main()
{
  string str = "geeks for geeks geeks quiz "
               "practice qa for";
  printFrequencies(str);
  return 0;
}
```

Output

```
(practice, 1)
(for, 2)
(qa, 1)
(quiz, 1)
(geeks, 3)
```

Recent Articles on unordered_map

This article is contributed by Utkarsh Trivedi. Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Previous Next

2. Dijkstra’s shortest path algorithm using set in STL
3. Dijkstra's Shortest Path Algorithm using priority_queue of STL
4. Prim's algorithm using priority_queue in STL
5. fill() and fill_n() functions in C++ STL
6. Count number of unique Triangles using STL | Set 1 (Using set)
7. Permutations of a given string using STL
8. Algorithm Library | C++ Magicians STL Algorithm
9. Array algorithms in C++ STL (all_of, any_of, none_of, copy_n and iota)
10. accumulate() and partial_sum() in C++ STL : Numeric header

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

Improved By : decoder_, omarfarag74, tejender, arorakashish0911, ahampriyanshu, bhartik021, anmol_choudhary, harsh_shokeen, demishassabis,

cpp-unordered_map-functions, STL, C++

Read Discuss Courses Practice Video

Improve Article Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Web Tutorials

Write an Article

Read Discuss Courses Practice Video

JavaScript

Bootstrap

ReactJS

NodeJS

Write Interview Experience

Internships

Video Internship

Start Your Coding Journey Now!