unordered_map in C++ STL

Difficulty Level : EasyLast Updated : 12 Feb, 2021

unordered_map is an associated container that stores elements formed by combination of key value and a mapped value. The key value is used to uniquely identify the element and mapped value is the content associated with the key. Both key and value can be of any type predefined or user-defined.
Internally unordered_map is implemented using Hash Table, the key provided to map are hashed into indices of hash table that is why performance of data structure depends on hash function a lot but on an average the cost of **search, insert and delete** from hash table is O(1).

CPP

CPP

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

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
GeeksforGeeks 10
Practice 20
```

unordered_map vs unordered_set :
In unordered_set, we have only key, no value, these are mainly used to see presence/absence in a set. For example, consider the problem of counting frequencies of individual words. We can't use unordered_set (or set) as we can't store counts.
unordered_map vs map :
map (like set) is an ordered sequence of unique keys whereas in unordered_map key can be stored in any order, so unordered.
Map is implemented as balanced tree structure that is why it is possible to maintain an order between the elements (by specific tree traversal). Time complexity of map operations is O(Log n) while for unordered_map, it is O(1) on average.
Methods on unordered_map
ot of function are available which work on unordered_map. most useful of them are – operator =, operator [], empty and size for capacity, begin and end for iterator, find and count for lookup, insert

and erase for modification.
The C++11 library also provides function to see internally used bucket count, bucket size and also used hash function and various hash policies but they are less useful in real application.
We can iterate over all elements of unordered_map using Iterator.
Initialization, indexing and iteration is shown in below sample code :

CPP

CPP

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

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 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
    umap.insert(make_pair("e", 2.718));

    string key = "PI";

    // If key not found in map iterator to end is returned
    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 stroes the value part
        cout << itr->first << "  " << itr->second << endl;
     }
}
```

Output:

```
Found PI

lambda not found

All Elements :
loge  1
e  2.718
log10  2.302
root3  1.732
PI  3.14
root2  1.414
```

practical problem based on unordered_map – given a string of words, find frequencies of 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 a C++ solution using unordered_map.

CPP

CPP

```cpp
// 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
    stringstream ss(str);  // Used for breaking words
    string word; // To store individual words
    while (ss >> word)
        wordFreq[word]++;

    // now iterating over word, freq pair and printing
    // them in <, > format
    unordered_map<string, int>:: iterator p;
    for (p = wordFreq.begin(); p != wordFreq.end(); p++)
```

```
        cout << "(" << p->first << ", " << p->second << ")\n";
}

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

Output:

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

Methods of unordered_map :

- 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 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.