**map:**

std::map  
  
You use map when:

* You need ordered data.
* You would have to print/access the data (in sorted order).
* You need predecessor/successor of elements.

**Internal implementation:**red black tree

unordered\_map:  
  
std::unordered\_map

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

**Internal Implementation:**

Every unordered\_map implementation stores a linked list to external nodes in the array of buckets. (chaining approach rather than open addressing)

**There are a set of reasons It uses chaining:**

insert and emplace members shall not affect the validity of iterators if (N+n) < z \* B, where N is the number of elements in the container prior to the insert operation, n is the number of elements inserted, B is the container’s bucket count, and z is the container’s maximum load factor.

(Now, unordered\_map also have bucket\_count and load\_factor)

**Now, unordered\_map main functions:**

**capacity**

**empty**

Test whether container is empty (public member function)

**size**

Return container size (public member function)

**max\_size**

Return maximum size (public member function)

**Iterators**

**begin**

Return iterator to beginning (public member function)

**end**

Return iterator to end (public member function)

**cbegin**

Return const\_iterator to beginning (public member function)

**cend**

Return const\_iterator to end (public member function)

**Element access**

**operator[]**

Access element (public member function )

**at**

Access element (public member function)

**Element lookup**

**find**

Get iterator to element (public member function)

**count**

Count elements with a specific key (public member function )

**equal\_range**

Get range of elements with specific key (public member function)

**Buckets**

**bucket\_count**

Return number of buckets (public member function)

**max\_bucket\_count**

Return maximum number of buckets (public member function)

**bucket\_size**

Return bucket size (public member type)

**bucket**

Locate element's bucket (public member function)

**Hash policy**

**load\_factor**

Return load factor (public member function)

**max\_load\_factor**

Get or set maximum load factor (public member function )

**rehash**

Set number of buckets (public member function )

**reserve**

Request a capacity change (public member function)

**Now, ordered\_map main functions:  
  
capacity**

**empty**

Test whether container is empty (public member function)

**size**

Return container size (public member function)

**max\_size**

Return maximum size (public member function)

**Iterators**

**begin**

Return iterator to beginning (public member function)

**end**

Return iterator to end (public member function)

**cbegin**

Return const\_iterator to beginning (public member function)

**cend**

Return const\_iterator to end (public member function)

**Element access**

**operator[]**

Access element (public member function )

**at**

Access element (public member function)

**Element lookup**

**find**

Get iterator to element (public member function)

**count**

Count elements with a specific key (public member function )

**equal\_range**

Get range of elements with specific key (public member function)

Now, what you don’t see the bucket and hash policy related functions (obviously) in map. Rather, you will find the following two functions:

**lower\_bound, upper\_bound**

**Multimap:**

A std::multimap is equal to a std::map, but your keys are not unique anymore. Therefore you can find a range of items instead of just find one unique item.

(we have std::multiset, too)