public member function

std::map::insert

<map>

C++98 C++11

```
single element (1)  pair<iterator,bool> insert (const value_type& val);
with hint (2)       iterator insert (iterator position, const value_type& val);
range (3)           template <class InputIterator>
                      void insert (InputIterator first, InputIterator last);
```

Insert elements

Extends the container by inserting new elements, effectively increasing the container size by the number of elements inserted.

Because element keys in a map are unique, the insertion operation checks whether each inserted element has a key equivalent to the one of an element already in the container, and if so, the element is not inserted, returning an iterator to this existing element (if the function returns a value).

For a similar container allowing for duplicate elements, see multimap.

An alternative way to insert elements in a map is by using member function map::operator[].

Internally, map containers keep all their elements sorted by their key following the criterion specified by its comparison object. The elements are always inserted in its respective position following this ordering.

The parameters determine how many elements are inserted and to which values they are initialized:

Parameters

`val`
Value to be copied to (or moved as) the inserted element.
Member type `value_type` is the type of the elements in the container, defined in map as `pair<const key_type,mapped_type>` (see map member types).

C++98 C++14

The template parameter P shall be a type convertible to `value_type`.
If P is instantiated as a reference type, the argument is copied.

`position`
Hint for the position where the element can be inserted.

C++98 C++11

The function optimizes its insertion time if *position* points to the element that will **precede** the inserted element.
Notice that this is just a hint and does not force the new element to be inserted at that position within the map container (the elements in a map always follow a specific order depending on their key).
Member types `iterator` and `const_iterator` are defined in map as bidirectional iterator types that point to elements.

`first, last`
Iterators specifying a range of elements. Copies of the elements in the range `[first,last)` are inserted in the container.
Notice that the range includes all the elements between *first* and *last*, including the element pointed by *first* but not the one pointed by *last*.
The function template argument `InputIterator` shall be an input iterator type that points to elements of a type from which `value_type` objects can be constructed.

`il`
An initializer_list object. Copies of these elements are inserted.
These objects are automatically constructed from *initializer list* declarators.
Member type `value_type` is the type of the elements contained in the container, defined in map as `pair<const key_type,mapped_type>` (see map member types).

Return value

The single element versions (1) return a pair, with its member `pair::first` set to an iterator pointing to either the newly inserted element or to the element with an equivalent key in the map. The `pair::second` element in the pair is set to `true` if a new element was inserted or `false` if an equivalent key already existed.

The versions with a hint (2) return an iterator pointing to either the newly inserted element or to the element that already had an equivalent key in the map.

Member type `iterator` is a bidirectional iterator type that points to elements.
pair is a class template declared in <utility> (see pair).

Example

```
1  // map::insert (C++98)
2  #include <iostream>
3  #include <map>
4
5  int main ()
6  {
7    std::map<char,int> mymap;
8
9    // first insert function version (single parameter):
10   mymap.insert ( std::pair<char,int>('a',100) );
11   mymap.insert ( std::pair<char,int>('z',200) );
12
13   std::pair<std::map<char,int>::iterator,bool> ret;
14   ret = mymap.insert ( std::pair<char,int>('z',500) );
15   if (ret.second==false) {
16     std::cout << "element 'z' already existed";
17     std::cout << " with a value of " << ret.first->second << '\n';
18   }
19
20   // second insert function version (with hint position):
21   std::map<char,int>::iterator it = mymap.begin();
22   mymap.insert (it, std::pair<char,int>('b',300));  // max efficiency inserting
     mymap.insert (it, std::pair<char,int>('c',400));  // no max efficiency inserting
```

```
23
24  // third insert function version (range insertion):
25  std::map<char,int> anothermap;
26  anothermap.insert(mymap.begin(),mymap.find('c'));
27
28  // showing contents:
29  std::cout << "mymap contains:\n";
30  for (it=mymap.begin(); it!=mymap.end(); ++it)
31    std::cout << it->first << " => " << it->second << '\n';
32
33  std::cout << "anothermap contains:\n";
34  for (it=anothermap.begin(); it!=anothermap.end(); ++it)
35    std::cout << it->first << " => " << it->second << '\n';
36
37  return 0;
38 }
39
```

Output:

```
element 'z' already existed with a value of 200
mymap contains:
a => 100
b => 300
c => 400
z => 200
anothermap contains:
a => 100
b => 300
```

Complexity

If a single element is inserted, logarithmic in size in general, but amortized constant if a hint is given and the *position* given is the optimal.

C++98 | C++11

If N elements are inserted, `N log(size+N)` in general, but linear in `size+N` if the elements are already sorted according to the same ordering criterion used by the container.

Iterator validity

No changes.

Data races

The container is modified.
Concurrently accessing existing elements is safe, although iterating ranges in the container is not.

Exception safety

If a single element is to be inserted, there are no changes in the container in case of exception (strong guarantee).
Otherwise, the container is guaranteed to end in a valid state (basic guarantee).
If allocator_traits::construct is not supported with the appropriate arguments for the element constructions, or if an invalid *position* is specified, it causes *undefined behavior*.

See also

map::operator[]	Access element (public member function)
map::find	Get iterator to element (public member function)
map::erase	Erase elements (public member function)