public member function

std::set::insert

<set>

C++98 C++11

single element (1)	`pair<iterator,bool> insert (const value_type& val);` `pair<iterator,bool> insert (value_type&& val);`
with hint (2)	`iterator insert (const_iterator position, const value_type& val);` `iterator insert (const_iterator position, value_type&& val);`
range (3)	`template <class InputIterator>` `  void insert (InputIterator first, InputIterator last);`
initializer list (4)	`void insert (initializer_list<value_type> il);`

Insert element

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

Because elements in a set are unique, the insertion operation checks whether each inserted element is equivalent to 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 multiset.

Internally, set containers keep all their elements sorted 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 (or moved) to the inserted elements.
Member type `value_type` is the type of the elements in the container, defined in set as an alias of its first template parameter (`T`).

`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 **follow** the inserted element (or to the end, if it would be the last).

Notice that this is just a hint and does not force the new element to be inserted at that position within the set container (the elements in a set always follow a specific order).
Member types `iterator` and `const_iterator` are defined in map as a bidirectional iterator type 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 in the container, defined in set as an alias of its first template parameter (`T`).

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 equivalent element already in the set. The `pair::second` element in the pair is set to `true` if a new element was inserted or `false` if an equivalent element 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 its same value in the set.

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

Example

```cpp
// set::insert (C++98)
#include <iostream>
#include <set>

int main ()
{
  std::set<int> myset;
  std::set<int>::iterator it;
  std::pair<std::set<int>::iterator,bool> ret;

  // set some initial values:
  for (int i=1; i<=5; ++i) myset.insert(i*10);    // set: 10 20 30 40 50

  ret = myset.insert(20);               // no new element inserted

  if (ret.second==false) it=ret.first;  // "it" now points to element 20

  myset.insert (it,25);                 // max efficiency inserting
  myset.insert (it,24);                 // max efficiency inserting
  myset.insert (it,26);                 // no max efficiency inserting

  int myints[]= {5,10,15};              // 10 already in set, not inserted
  myset.insert (myints,myints+3);

  std::cout << "myset contains:";
  for (it=myset.begin(); it!=myset.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

Output:

```
myset contains: 5 10 15 20 24 25 26 30 40 50
```

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`).
Implementations may optimize if the range is already sorted.

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

set::erase	Erase elements (public member function)
set::find	Get iterator to element (public member function)