class template

<set>

**std::set**

template < class T, // set::key\_type/value\_type

class Compare = less<T>, // set::key\_compare/value\_compare

class Alloc = allocator<T> // set::allocator\_type

> class set;

Set

Sets are containers that store unique elements following a specific order.  
  
In a set, the value of an element also identifies it (the value is itself the *key*, of type T), and each value must be unique. The value of the elements in a set cannot be modified once in the container (the elements are always const), but they can be inserted or removed from the container.  
  
Internally, the elements in a set are always sorted following a specific *strict weak ordering* criterion indicated by its internal [comparison object](http://www.cplusplus.com/set::key_comp) (of type Compare).  
  
set containers are generally slower than [unordered\_set](http://www.cplusplus.com/unordered_set) containers to access individual elements by their *key*, but they allow the direct iteration on subsets based on their order.  
  
Sets are typically implemented as *binary search trees*.

**Container properties**

Associative

Elements in associative containers are referenced by their *key* and not by their absolute position in the container.

Ordered

The elements in the container follow a strict order at all times. All inserted elements are given a position in this order.

Set

The value of an element is also the *key* used to identify it.

Unique keys

No two elements in the container can have equivalent *keys*.

Allocator-aware

The container uses an allocator object to dynamically handle its storage needs.

**Template parameters**

T

Type of the elements. Each element in a set container is also uniquely identified by this value (each value is itself also the element's key).  
Aliased as member types set::key\_type and set::value\_type.

Compare

A binary predicate that takes two arguments of the same type as the elements and returns a bool. The expression comp(a,b), where *comp* is an object of this type and *a* and *b* are key values, shall return true if *a* is considered to go before *b* in the *strict weak ordering* the function defines.  
The set object uses this expression to determine both the order the elements follow in the container and whether two element keys are equivalent (by comparing them reflexively: they are equivalent if !comp(a,b) && !comp(b,a)). No two elements in a set container can be equivalent.  
This can be a function pointer or a function object (see [constructor](http://www.cplusplus.com/set::set) for an example). This defaults to [less](http://www.cplusplus.com/less)<T>, which returns the same as applying the *less-than operator* (a<b).  
Aliased as member types set::key\_compare and set::value\_compare.

Alloc

Type of the allocator object used to define the storage allocation model. By default, the [allocator](http://www.cplusplus.com/allocator) class template is used, which defines the simplest memory allocation model and is value-independent.  
Aliased as member type set::allocator\_type.

**Member types**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |  |
| --- | --- | --- |
| **member type** | **definition** | **notes** |
| key\_type | The first template parameter (T) |  |
| value\_type | The first template parameter (T) |  |
| key\_compare | The second template parameter (Compare) | defaults to: [less](http://www.cplusplus.com/less)<key\_type> |
| value\_compare | The second template parameter (Compare) | defaults to: [less](http://www.cplusplus.com/less)<value\_type> |
| allocator\_type | The third template parameter (Alloc) | defaults to: [allocator](http://www.cplusplus.com/allocator)<value\_type> |
| reference | allocator\_type::reference | for the default [allocator](http://www.cplusplus.com/allocator): value\_type& |
| const\_reference | allocator\_type::const\_reference | for the default [allocator](http://www.cplusplus.com/allocator): const value\_type& |
| pointer | allocator\_type::pointer | for the default [allocator](http://www.cplusplus.com/allocator): value\_type\* |
| const\_pointer | allocator\_type::const\_pointer | for the default [allocator](http://www.cplusplus.com/allocator): const value\_type\* |
| iterator | a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) to value\_type | convertible to const\_iterator |
| const\_iterator | a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) to const value\_type |  |
| reverse\_iterator | [reverse\_iterator](http://www.cplusplus.com/reverse_iterator)<iterator> |  |
| const\_reverse\_iterator | [reverse\_iterator](http://www.cplusplus.com/reverse_iterator)<const\_iterator> |  |
| difference\_type | a signed integral type, identical to: iterator\_traits<iterator>::difference\_type | usually the same as [ptrdiff\_t](http://www.cplusplus.com/ptrdiff_t) |
| size\_type | an unsigned integral type that can represent any non-negative value of difference\_type | usually the same as [size\_t](http://www.cplusplus.com/size_t) |

**Member functions**

[**(constructor)**](http://www.cplusplus.com/reference/set/set/set/)

Construct set (public member function )

[**(destructor)**](http://www.cplusplus.com/reference/set/set/%7Eset/)

Set destructor (public member function )

[**operator=**](http://www.cplusplus.com/reference/set/set/operator=/)

Copy container content (public member function )

**Iterators**:

[**begin**](http://www.cplusplus.com/reference/set/set/begin/)

Return iterator to beginning (public member function )

[**end**](http://www.cplusplus.com/reference/set/set/end/)

Return iterator to end (public member function )

[**rbegin**](http://www.cplusplus.com/reference/set/set/rbegin/)

Return reverse iterator to reverse beginning (public member function )

[**rend**](http://www.cplusplus.com/reference/set/set/rend/)

Return reverse iterator to reverse end (public member function )

[**cbegin**](http://www.cplusplus.com/reference/set/set/cbegin/)

Return const\_iterator to beginning (public member function )

[**cend**](http://www.cplusplus.com/reference/set/set/cend/)

Return const\_iterator to end (public member function )

[**crbegin**](http://www.cplusplus.com/reference/set/set/crbegin/)

Return const\_reverse\_iterator to reverse beginning (public member function )

[**crend**](http://www.cplusplus.com/reference/set/set/crend/)

Return const\_reverse\_iterator to reverse end (public member function )

**Capacity**:

[**empty**](http://www.cplusplus.com/reference/set/set/empty/)

Test whether container is empty (public member function )

[**size**](http://www.cplusplus.com/reference/set/set/size/)

Return container size (public member function )

[**max\_size**](http://www.cplusplus.com/reference/set/set/max_size/)

Return maximum size (public member function )

**Modifiers**:

[**insert**](http://www.cplusplus.com/reference/set/set/insert/)

Insert element (public member function )

[**erase**](http://www.cplusplus.com/reference/set/set/erase/)

Erase elements (public member function )

[**swap**](http://www.cplusplus.com/reference/set/set/swap/)

Swap content (public member function )

[**clear**](http://www.cplusplus.com/reference/set/set/clear/)

Clear content (public member function )

[**emplace**](http://www.cplusplus.com/reference/set/set/emplace/)

Construct and insert element (public member function )

[**emplace\_hint**](http://www.cplusplus.com/reference/set/set/emplace_hint/)

Construct and insert element with hint (public member function )

**Observers**:

[**key\_comp**](http://www.cplusplus.com/reference/set/set/key_comp/)

Return comparison object (public member function )

[**value\_comp**](http://www.cplusplus.com/reference/set/set/value_comp/)

Return comparison object (public member function )

**Operations**:

[**find**](http://www.cplusplus.com/reference/set/set/find/)

Get iterator to element (public member function )

[**count**](http://www.cplusplus.com/reference/set/set/count/)

Count elements with a specific value (public member function )

[**lower\_bound**](http://www.cplusplus.com/reference/set/set/lower_bound/)

Return iterator to lower bound (public member function )

[**upper\_bound**](http://www.cplusplus.com/reference/set/set/upper_bound/)

Return iterator to upper bound (public member function )

[**equal\_range**](http://www.cplusplus.com/reference/set/set/equal_range/)

Get range of equal elements (public member function )

**Allocator**:

[**get\_allocator**](http://www.cplusplus.com/reference/set/set/get_allocator/)

Get allocator (public member function )

public member function

<set>

**std::**[**set**](http://www.cplusplus.com/reference/set/set/)**::set**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **empty (1)** | explicit set (const key\_compare& comp = key\_compare(),  const allocator\_type& alloc = allocator\_type()); |
| **range (2)** | template <class InputIterator>  set (InputIterator first, InputIterator last,  const key\_compare& comp = key\_compare(),  const allocator\_type& alloc = allocator\_type()); |
| **copy (3)** | set (const set& x); |

Construct set

Constructs a [set](http://www.cplusplus.com/set) container object, initializing its contents depending on the constructor version used:

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

(1) empty container constructor (default constructor)

Constructs an [empty](http://www.cplusplus.com/set::empty) container, with no elements.

(2) range constructor

Constructs a container with as many elements as the range [first,last), with each element constructed from its corresponding element in that range.

(3) copy constructor

Constructs a container with a copy of each of the elements in *x*.

The container keeps an internal copy of *alloc* and *comp*, which are used to allocate storage and to sort the elements throughout its lifetime.  
The copy constructor *(3)* creates a container that keeps and uses copies of *x*'s [allocator](http://www.cplusplus.com/set::get_allocator) and [comparison object](http://www.cplusplus.com/set::key_comp).  
  
The storage for the elements is allocated using this [internal allocator](http://www.cplusplus.com/set::get_allocator).

The elements are sorted according to the [comparison object](http://www.cplusplus.com/set::key_comp). If more than one equivalent element is passed to the constructor, only the first one is preserved.

**Parameters**

comp

Binary predicate that, taking two values of the same type of those contained in the [set](http://www.cplusplus.com/set), returns true if the first argument goes before the second argument in the *strict weak ordering* it defines, and false otherwise.  
This shall be a function pointer or a function object.  
Member type key\_compare is the internal comparison object type used by the container, defined in [set](http://www.cplusplus.com/set) as an alias of its second template parameter (Compare).  
If key\_compare uses the default [less](http://www.cplusplus.com/less) (which has no state), this parameter is not relevant.

alloc

Allocator object.  
The container keeps and uses an internal copy of this allocator.  
Member type allocator\_type is the internal allocator type used by the container, defined in [set](http://www.cplusplus.com/set) as an alias of its third template parameter (Alloc).  
If allocator\_type is an instantiation of the default [allocator](http://www.cplusplus.com/allocator) (which has no state), this parameter is not relevant.

first, last

[Input iterators](http://www.cplusplus.com/InputIterator) to the initial and final positions in a range. The range used is [first,last), which includes all the elements between *first* and *last*, including the element pointed by *first* but not the element pointed by *last*.  
The function template argument InputIterator shall be an [input iterator](http://www.cplusplus.com/InputIterator) type that points to elements of a type from which value\_type objects can be constructed.

x

Another [set](http://www.cplusplus.com/set) object of the same type (with the same class template arguments T, Compare and Alloc), whose contents are either copied or acquired.

il

An [initializer\_list](http://www.cplusplus.com/initializer_list) object.  
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](http://www.cplusplus.com/set) as an alias of its first template parameter (T).

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 | *// constructing sets*  *#include <iostream>*  *#include <set>*  *bool* fncomp (*int* lhs, *int* rhs) {*return* lhs<rhs;}  *struct* classcomp {  *bool* *operator*() (*const* *int*& lhs, *const* *int*& rhs) *const*  {*return* lhs<rhs;}  };  *int* main ()  {  std::set<*int*> first; *// empty set of ints*  *int* myints[]= {10,20,30,40,50};  std::set<*int*> second (myints,myints+5); *// range*  std::set<*int*> third (second); *// a copy of second*  std::set<*int*> fourth (second.begin(), second.end()); *// iterator ctor.*  std::set<*int*,classcomp> fifth; *// class as Compare*  *bool*(\*fn\_pt)(*int*,*int*) = fncomp;  std::set<*int*,*bool*(\*)(*int*,*int*)> sixth (fn\_pt); *// function pointer as Compare*  *return* 0;  } |

The code does not produce any output, but demonstrates some ways in which a [set](http://www.cplusplus.com/set) container can be constructed.

**Complexity**

Constant for the *empty constructors (1)*, and for the *move constructors (4)* (unless *alloc* is different from *x*'s allocator).  
For all other cases, linear in the distance between the iterators (copy constructions) if the elements are already sorted according to the same criterion. For unsorted sequences, linearithmic (N\*logN) in that distance (sorting,copy constructions).

**Iterator validity**

The *move constructors* (4), invalidate all iterators, pointers and references related to *x* if the elements are moved.

**Data races**

All copied elements are accessed.  
The *move constructors (4)* modify *x*.

**Exception safety**

**Strong guarantee:** no effects in case an exception is thrown.  
If [allocator\_traits::construct](http://www.cplusplus.com/allocator_traits::construct) is not supported with the appropriate arguments for the element constructions, or if the range specified by [first,last) is not valid, it causes *undefined behavior*.

public member function

<set>

**std::**[**set**](http://www.cplusplus.com/reference/set/set/)**::~set**

~set();

Set destructor

Destroys the container object.

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

This destroys all container elements, and deallocates all the storage capacity allocated by the [set](http://www.cplusplus.com/set) container using its [allocator](http://www.cplusplus.com/set::get_allocator).

**Complexity**

Linear in [set::size](http://www.cplusplus.com/set::size) (destructors).

**Iterator validity**

All iterators, pointers and references are invalidated.

**Data races**

The container and all its elements are modified.

**Exception safety**

**No-throw guarantee:** never throws exceptions.

public member function

<set>

**std::**[**set**](http://www.cplusplus.com/reference/set/set/)**::operator=**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **copy (1)** | set& operator= (const set& x); |

Copy container content

Assigns new contents to the container, replacing its current content.

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

Copies all the elements from *x* into the container, changing its [size](http://www.cplusplus.com/set::size) accordingly.  
  
The container preserves its [current allocator](http://www.cplusplus.com/set::get_allocator), which is used to allocate additional storage if needed.

The elements stored in the container before the call are either assigned to or destroyed.

**Parameters**

x

A [set](http://www.cplusplus.com/set) object of the same type (i.e., with the same template parameters, T, Compare and Alloc).

il

An [initializer\_list](http://www.cplusplus.com/initializer_list) object. The compiler will automatically construct such objects from *initializer list* declarators.  
Member type value\_type is the type of the elements in the container, defined in [set](http://www.cplusplus.com/set) as an alias of its first template parameter (T).

**Return value**

\*this

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 | *// assignment operator with sets*  *#include <iostream>*  *#include <set>*  *int* main ()  {  *int* myints[]={ 12,82,37,64,15 };  std::set<*int*> first (myints,myints+5); *// set with 5 ints*  std::set<*int*> second; *// empty set*  second = first; *// now second contains the 5 ints*  first = std::set<*int*>(); *// and first is empty*  std::cout << "Size of first: " << *int* (first.size()) << '\n';  std::cout << "Size of second: " << *int* (second.size()) << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| Size of first: 0  Size of second: 5 |

**Complexity**

For the *copy assignment (1)*: Linear in sizes (destructions, copies).  
For the *move assignment (2)*: Linear in current container [size](http://www.cplusplus.com/set::size) (destructions).\*   
For the *initializer list assignment (3)*: Up to logarithmic in sizes (destructions, move-assignments) -- linear if *il* is already sorted.  
\* Additional complexity for assignments if allocators do not [*propagate*](http://www.cplusplus.com/allocator_traits#types).

**Iterator validity**

All iterators, references and pointers related to this container are invalidated.  
  
In the *move assignment*, iterators, pointers and references referring to elements in *x* are also invalidated.

**Data races**

All copied elements are accessed.  
The *move assignment (2)* modifies *x*.  
The container and all its elements are modified.

**Exception safety**

**Basic guarantee:** if an exception is thrown, the container is in a valid state.  
If [allocator\_traits::construct](http://www.cplusplus.com/allocator_traits::construct) is not supported with the appropriate arguments for the element constructions, or if value\_type is not [copy assignable](http://www.cplusplus.com/is_copy_assignable) (or [move assignable](http://www.cplusplus.com/is_move_assignable) for *(2)*), it causes *undefined behavior*.

public member function

<set>

**std::**[**set**](http://www.cplusplus.com/reference/set/set/)**::begin**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

iterator begin();

const\_iterator begin() const;

Return iterator to beginning

Returns an iterator referring to the first element in the [set](http://www.cplusplus.com/set) container.  
  
Because [set](http://www.cplusplus.com/set) containers keep their elements ordered at all times, begin points to the element that goes first following the container's [sorting criterion](http://www.cplusplus.com/set::key_comp).  
  
If the container is [empty](http://www.cplusplus.com/set::empty), the returned iterator value shall not be dereferenced.

**Parameters**

none

**Return Value**

An iterator to the first element in the container.  
  
If the [set](http://www.cplusplus.com/set) object is const-qualified, the function returns a const\_iterator. Otherwise, it returns an iterator.  
  
Member types iterator and const\_iterator are [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) types pointing to elements.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 | *// set::begin/end*  *#include <iostream>*  *#include <set>*  *int* main ()  {  *int* myints[] = {75,23,65,42,13};  std::set<*int*> myset (myints,myints+5);  std::cout << "myset contains:";  *for* (std::set<*int*>::iterator it=myset.begin(); it!=myset.end(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| myset contains: 13 23 42 65 75 |

**Complexity**

Constant.

**Iterator validity**

No changes.

**Data races**

The container is accessed (neither the const nor the non-const versions modify the container).  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<set>

**std::**[**set**](http://www.cplusplus.com/reference/set/set/)**::end**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

iterator end();

const\_iterator end() const;

Return iterator to end

Returns an iterator referring to the *past-the-end* element in the [set](http://www.cplusplus.com/set) container.  
  
The *past-the-end* element is the theoretical element that would follow the last element in the [set](http://www.cplusplus.com/set) container. It does not point to any element, and thus shall not be dereferenced.  
  
Because the ranges used by functions of the standard library do not include the element pointed by their closing iterator, this function is often used in combination with [set::begin](http://www.cplusplus.com/set::begin) to specify a range including all the elements in the container.  
  
If the container is [empty](http://www.cplusplus.com/set::empty), this function returns the same as [set::begin](http://www.cplusplus.com/set::begin).

**Parameters**

none

**Return Value**

An iterator to the *past-the-end* element in the container.  
  
If the [set](http://www.cplusplus.com/set) object is const-qualified, the function returns a const\_iterator. Otherwise, it returns an iterator.  
  
Member types iterator and const\_iterator are [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) types pointing to elements.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 | *// set::begin/end*  *#include <iostream>*  *#include <set>*  *int* main ()  {  *int* myints[] = {75,23,65,42,13};  std::set<*int*> myset (myints,myints+5);  std::cout << "myset contains:";  *for* (std::set<*int*>::iterator it=myset.begin(); it!=myset.end(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| myset contains: 13 23 42 65 75 |

**Complexity**

Constant.

**Iterator validity**

No changes.

**Data races**

The container is accessed (neither the const nor the non-const versions modify the container).  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<set>

**std::**[**set**](http://www.cplusplus.com/reference/set/set/)**::rbegin**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

reverse\_iterator rbegin();

const\_reverse\_iterator rbegin() const;

Return reverse iterator to reverse beginning

Returns a *reverse iterator* pointing to the last element in the container (i.e., its *reverse beginning*).  
  
*Reverse iterators* iterate backwards: increasing them moves them towards the beginning of the container.  
  
rbegin points to the element preceding the one that would be pointed to by member [end](http://www.cplusplus.com/set::end).

**Parameters**

none

**Return Value**

A reverse iterator to the *reverse beginning* of the sequence container.  
  
If the [set](http://www.cplusplus.com/set) object is const-qualified, the function returns a const\_reverse\_iterator. Otherwise, it returns a reverse\_iterator.  
  
Member types reverse\_iterator and const\_reverse\_iterator are reverse [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) types pointing to elements. See [set member types](http://www.cplusplus.com/set#types).

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 | *// set::rbegin/rend*  *#include <iostream>*  *#include <set>*  *int* main ()  {  *int* myints[] = {21,64,17,78,49};  std::set<*int*> myset (myints,myints+5);  std::set<*int*>::reverse\_iterator rit;  std::cout << "myset contains:";  *for* (rit=myset.rbegin(); rit != myset.rend(); ++rit)  std::cout << ' ' << \*rit;  std::cout << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| myset contains: 78 64 49 21 17 |

**Complexity**

Constant.

**Iterator validity**

No changes.

**Data races**

The container is accessed (neither the const nor the non-const versions modify the container).  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<set>

**std::**[**set**](http://www.cplusplus.com/reference/set/set/)**::rend**

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

reverse\_iterator rend();

const\_reverse\_iterator rend() const;

Return reverse iterator to reverse end

Returns a *reverse iterator* pointing to the theoretical element right before the first element in the [set](http://www.cplusplus.com/set) container (which is considered its *reverse end*).  
  
The range between [set::rbegin](http://www.cplusplus.com/set::rbegin) and set::rend contains all the elements of the container (in reverse order).

**Parameters**

none

**Return Value**

A reverse iterator to the *reverse end* of the sequence container.  
  
If the [set](http://www.cplusplus.com/set) object is const-qualified, the function returns a const\_reverse\_iterator. Otherwise, it returns a reverse\_iterator.  
  
Member types reverse\_iterator and const\_reverse\_iterator are reverse [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) types pointing to elements. See [set member types](http://www.cplusplus.com/set#types).

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 | *// set::rbegin/rend*  *#include <iostream>*  *#include <set>*  *int* main ()  {  *int* myints[] = {21,64,17,78,49};  std::set<*int*> myset (myints,myints+5);  std::set<*int*>::reverse\_iterator rit;  std::cout << "myset contains:";  *for* (rit=myset.rbegin(); rit != myset.rend(); ++rit)  std::cout << ' ' << \*rit;  std::cout << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| myset contains: 78 64 49 21 17 |

**Complexity**

Constant.

**Iterator validity**

No changes.

**Data races**

The container is accessed (neither the const nor the non-const versions modify the container).  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<set>

**std::**[**set**](http://www.cplusplus.com/reference/set/set/)**::cbegin**

const\_iterator cbegin() const noexcept;

Return const\_iterator to beginning

Returns a const\_iterator pointing to the first element in the container.  
  
All iterators in [set](http://www.cplusplus.com/set) containers are *constant iterators* (including both const\_iterator and iterator member types). These cannot be used to modify the contents they point to, but can be increased and decreased normally (unless they are themselves also const).  
  
If the container is [empty](http://www.cplusplus.com/set::empty), the returned iterator value shall not be dereferenced.

**Parameters**

none

**Return Value**

A const\_iterator to the beginning of the sequence.  
  
Member type const\_iterator is a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) type that points to const elements.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 | *// set::cbegin/cend*  *#include <iostream>*  *#include <set>*  *int* main ()  {  std::set<*int*> myset = {50,20,60,10,25};  std::cout << "myset contains:";  *for* (*auto* it=myset.cbegin(); it != myset.cend(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| myset contains: 10 20 25 50 60 |

**Complexity**

Constant.

**Iterator validity**

No changes.

**Data races**

The container is accessed.  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<set>

**std::**[**set**](http://www.cplusplus.com/reference/set/set/)**::cend**

const\_iterator cend() const noexcept;

Return const\_iterator to end

Returns a const\_iterator pointing to the *past-the-end* element in the container.  
  
All iterators in [set](http://www.cplusplus.com/set) containers are *constant iterators* (including both const\_iterator and iterator member types). These cannot be used to modify the contents they point to, but can be increased and decreased normally (unless they are themselves also const).

**Parameters**

none

**Return Value**

A const\_iterator to the element past the end of the sequence.  
  
Member type const\_iterator is a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) type that points to a const element.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 | *// set::cbegin/cend*  *#include <iostream>*  *#include <set>*  *int* main ()  {  std::set<*int*> myset = {50,20,60,10,25};  std::cout << "myset contains:";  *for* (*auto* it=myset.cbegin(); it != myset.cend(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| myset contains: 10 20 25 50 60 |

**Complexity**

Constant.

**Iterator validity**

No changes.

**Data races**

The container is accessed.  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<set>

**std::**[**set**](http://www.cplusplus.com/reference/set/set/)**::crbegin**

const\_reverse\_iterator crbegin() const noexcept;

Return const\_reverse\_iterator to reverse beginning

Returns a const\_reverse\_iterator pointing to the last element in the container (i.e., its *reverse beginning*).

**Parameters**

none

**Return Value**

A const\_reverse\_iterator to the *reverse beginning* of the sequence.  
  
Member type const\_reverse\_iterator is a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) type that points to a const element.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 | *// set::crbegin/crend*  *#include <iostream>*  *#include <set>*  *int* main ()  {  std::set<*int*> myset = {50,20,60,10,25};  std::cout << "myset backwards:";  *for* (*auto* rit=myset.crbegin(); rit != myset.crend(); ++rit)  std::cout << ' ' << \*rit;  std::cout << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| myset backwards: 60 50 25 20 10 |

**Complexity**

Constant.

**Iterator validity**

No changes.

**Data races**

The container is accessed.  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<set>

**std::**[**set**](http://www.cplusplus.com/reference/set/set/)**::crend**

const\_reverse\_iterator crend() const noexcept;

Return const\_reverse\_iterator to reverse end

Returns a const\_reverse\_iterator pointing to the element that would theoretically precede the first element in the container (which is considered its *reverse end*).

**Parameters**

none

**Return Value**

A const\_reverse\_iterator to the *reverse end* of the sequence.  
  
Member type const\_reverse\_iterator is a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) type that points to a const element.

**Example**

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 | *// set::crbegin/crend*  *#include <iostream>*  *#include <set>*  *int* main ()  {  std::set<*int*> myset = {50,20,60,10,25};  std::cout << "myset backwards:";  *for* (*auto* rit=myset.crbegin(); rit != myset.crend(); ++rit)  std::cout << ' ' << \*rit;  std::cout << '\n';  *return* 0;  } |

Output:

|  |
| --- |
| myset backwards: 60 50 25 20 10 |

**Complexity**

Constant.

**Iterator validity**

No changes.

**Data races**

The container is accessed.  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

**Exception safety**

**No-throw guarantee:** this member function never throws exceptions.  
The copy construction or assignment of the returned iterator is also guaranteed to never throw.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::empty

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

bool empty() const;

Test whether container is empty

Returns whether the [set](http://www.cplusplus.com/set) container is empty (i.e. whether its [size](http://www.cplusplus.com/set::size) is 0).  
  
This function does not modify the container in any way. To clear the content of a [set](http://www.cplusplus.com/set) container, see [set::clear](http://www.cplusplus.com/set::clear).

### Parameters

none

### Return Value

true if the container [size](http://www.cplusplus.com/set::size) is 0, false otherwise.

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 | // set::empty  #include <iostream>  #include <set>  int main ()  {  std::set<int> myset;  myset.insert(20);  myset.insert(30);  myset.insert(10);  std::cout << "myset contains:";  while (!myset.empty())  {  std::cout << ' ' << \*myset.begin();  myset.erase(myset.begin());  }  std::cout << '\n';  return 0;  } |

Output:

|  |
| --- |
| myset contains: 10 20 30 |

### Complexity

Constant.

### Iterator validity

No changes.

### Data races

The container is accessed.  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

### Exception safety

**No-throw guarantee:** this member function never throws exceptions.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::size

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

size\_type size() const;

Return container size

Returns the number of elements in the [set](http://www.cplusplus.com/set) container.

### Parameters

none

### Return Value

The number of elements in the container.  
  
Member type size\_type is an unsigned integral type.

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 | // set::size  #include <iostream>  #include <set>  int main ()  {  std::set<int> myints;  std::cout << "0. size: " << myints.size() << '\n';  for (int i=0; i<10; ++i) myints.insert(i);  std::cout << "1. size: " << myints.size() << '\n';  myints.insert (100);  std::cout << "2. size: " << myints.size() << '\n';  myints.erase(5);  std::cout << "3. size: " << myints.size() << '\n';  return 0;  } |

Output:

|  |
| --- |
| 0. size: 0  1. size: 10  2. size: 11  3. size: 10 |

### Complexity

Constant.

### Iterator validity

No changes.

### Data races

The container is accessed.  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

### Exception safety

**No-throw guarantee:** this member function never throws exceptions.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::max\_size

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

size\_type max\_size() const;

Return maximum size

Returns the maximum number of elements that the [set](http://www.cplusplus.com/set) container can hold.  
  
This is the maximum potential [size](http://www.cplusplus.com/set::size) the container can reach due to known system or library implementation limitations, but the container is by no means guaranteed to be able to reach that size: it can still fail to allocate storage at any point before that size is reached.

### Parameters

none

### Return Value

The maximum number of elements a [set](http://www.cplusplus.com/set) container can hold as content.  
  
Member type size\_type is an unsigned integral type.

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 | // set::max\_size  #include <iostream>  #include <set>  int main ()  {  int i;  std::set<int> myset;  if (myset.max\_size()>1000)  {  for (i=0; i<1000; i++) myset.insert(i);  std::cout << "The set contains 1000 elements.\n";  }  else std::cout << "The set could not hold 1000 elements.\n";  return 0;  } |

Here, member max\_size is used to check beforehand whether the set will allow for 1000 elements to be inserted.

### Complexity

Constant.

### Iterator validity

No changes.

### Data races

The container is accessed.  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

### Exception safety

**No-throw guarantee:** this member function never throws exceptions.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::insert

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **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 element

Extends the container by inserting new elements, effectively increasing the container [size](http://www.cplusplus.com/set::size) by the number of elements inserted.  
  
Because elements in a [set](http://www.cplusplus.com/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](http://www.cplusplus.com/multiset).  
  
Internally, [set](http://www.cplusplus.com/set) containers keep all their elements sorted following the criterion specified by its [comparison object](http://www.cplusplus.com/set::key_comp). 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](http://www.cplusplus.com/set) as an alias of its first template parameter (T).

position

Hint for the position where the element can be inserted.

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

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 [set](http://www.cplusplus.com/set) container (the elements in a [set](http://www.cplusplus.com/set) always follow a specific order).  
Member types iterator and const\_iterator are defined in [map](http://www.cplusplus.com/map) as a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) 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](http://www.cplusplus.com/InputIterator) type that points to elements of a type from which value\_type objects can be constructed.

il

An [initializer\_list](http://www.cplusplus.com/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](http://www.cplusplus.com/set) as an alias of its first template parameter (T).

### Return value

The single element versions (1) return a [pair](http://www.cplusplus.com/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](http://www.cplusplus.com/set). The pair::second element in the [pair](http://www.cplusplus.com/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](http://www.cplusplus.com/set).  
  
Member type iterator is a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) type that points to elements.  
[pair](http://www.cplusplus.com/pair) is a class template declared in [<utility>](http://www.cplusplus.com/%3Cutility%3E) (see [pair](http://www.cplusplus.com/pair)).

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 | // 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](http://www.cplusplus.com/set::size) in general, but amortized constant if a hint is given and the *position* given is the optimal.

* [C++98](javascript:switch3.select(1))
* [C++11](javascript:switch3.select(2))

If N elements are inserted, Nlog([size](http://www.cplusplus.com/set::size)+N) in general, but linear in [size](http://www.cplusplus.com/set::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](http://www.cplusplus.com/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*.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::erase

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

|  |  |
| --- | --- |
| **(1)** | void erase (iterator position); |
| **(2)** | size\_type erase (const value\_type& val); |
| **(3)** | void erase (iterator first, iterator last); |

Erase elements

Removes from the [set](http://www.cplusplus.com/set) container either a single element or a range of elements ([first,last)).  
  
This effectively reduces the container [size](http://www.cplusplus.com/set::size) by the number of elements removed, which are destroyed.

### Parameters

position

Iterator pointing to a single element to be removed from the [set](http://www.cplusplus.com/set).  
Member types iterator and const\_iterator are [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) types that point to elements.

val

Value to be removed from the [set](http://www.cplusplus.com/set).  
Member type value\_type is the type of the elements in the container, defined in [set](http://www.cplusplus.com/set) as an alias of its first template parameter (T).

first, last

Iterators specifying a range within the [set](http://www.cplusplus.com/set) container to be removed: [first,last). i.e., the range includes all the elements between *first* and *last*, including the element pointed by *first* but not the one pointed by *last*.  
Member types iterator and const\_iterator are [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) types that point to elements.

### Return value

For the value-based version (2), the function returns the number of elements erased, which in [set](http://www.cplusplus.com/set) containers is at most *1*.  
  
Member type size\_type is an unsigned integral type.

* [C++98](javascript:switch2.select(1))
* [C++11](javascript:switch2.select(2))

The other versions return no value.

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 | // erasing from set  #include <iostream>  #include <set>  int main ()  {  std::set<int> myset;  std::set<int>::iterator it;  // insert some values:  for (int i=1; i<10; i++) myset.insert(i\*10); // 10 20 30 40 50 60 70 80 90  it = myset.begin();  ++it; // "it" points now to 20  myset.erase (it);  myset.erase (40);  it = myset.find (60);  myset.erase (it, myset.end());  std::cout << "myset contains:";  for (it=myset.begin(); it!=myset.end(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  return 0;  } |

Output:

|  |
| --- |
| myset contains: 10 30 50 |

### Complexity

For the first version (erase(position)), amortized constant.  
For the second version (erase(val)), logarithmic in container [size](http://www.cplusplus.com/set::size).  
For the last version (erase(first,last)), linear in the distance between *first* and *last*.

### Iterator validity

Iterators, pointers and references referring to elements removed by the function are invalidated.  
All other iterators, pointers and references keep their validity.

### Data races

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

### Exception safety

Unless the container's [comparison object](http://www.cplusplus.com/set::key_comp) throws, this function never throws exceptions (no-throw guarantee).  
Otherwise, if a single element is to be removed, 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 an invalid *position* or range is specified, it causes *undefined behavior*.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::swap

void swap (set& x);

Swap content

Exchanges the content of the container by the content of *x*, which is another [set](http://www.cplusplus.com/set) of the same type. Sizes may differ.  
  
After the call to this member function, the elements in this container are those which were in *x* before the call, and the elements of *x* are those which were in this. All iterators, references and pointers remain valid for the swapped objects.  
  
Notice that a non-member function exists with the same name, [swap](http://www.cplusplus.com/set:swap), overloading that algorithm with an optimization that behaves like this member function.

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

Whether the internal container [allocators](http://www.cplusplus.com/set::get_allocator) and [comparison objects](http://www.cplusplus.com/set::key_comp) are swapped is undefined.

### Parameters

x

Another [set](http://www.cplusplus.com/set) container of the same type as this (i.e., with the same template parameters, T, Compare and Alloc) whose content is swapped with that of this container.

### Return value

none

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 | // swap sets  #include <iostream>  #include <set>  main ()  {  int myints[]={12,75,10,32,20,25};  std::set<int> first (myints,myints+3); // 10,12,75  std::set<int> second (myints+3,myints+6); // 20,25,32  first.swap(second);  std::cout << "first contains:";  for (std::set<int>::iterator it=first.begin(); it!=first.end(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  std::cout << "second contains:";  for (std::set<int>::iterator it=second.begin(); it!=second.end(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  return 0;  } |

Output:

|  |
| --- |
| first contains: 20 25 32  second contains: 10 12 75 |

### Complexity

Constant.

### Iterator validity

All iterators, pointers and references referring to elements in both containers remain valid, but now are referring to elements in the other container, and iterate in it.  
Note that the *end iterators* do not refer to elements and may be invalidated.

### Data races

Both the container and *x* are modified.  
No contained elements are accessed by the call (although see *iterator validity* above).

### Exception safety

If the allocators in both containers compare equal, or if their [allocator traits](http://www.cplusplus.com/allocator_traits) indicate that the allocators shall [propagate](http://www.cplusplus.com/allocator_traits#types), the function never throws exceptions (no-throw guarantee).  
Otherwise, it causes *undefined behavior*.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::clear

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

void clear();

Clear content

Removes all elements from the [set](http://www.cplusplus.com/set) container (which are destroyed), leaving the container with a [size](http://www.cplusplus.com/set::size) of 0.

### Parameters

none

### Return value

none

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 | // set::clear  #include <iostream>  #include <set>  int main ()  {  std::set<int> myset;  myset.insert (100);  myset.insert (200);  myset.insert (300);  std::cout << "myset contains:";  for (std::set<int>::iterator it=myset.begin(); it!=myset.end(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  myset.clear();  myset.insert (1101);  myset.insert (2202);  std::cout << "myset contains:";  for (std::set<int>::iterator it=myset.begin(); it!=myset.end(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  return 0;  } |

Output:

|  |
| --- |
| myset contains: 100 200 300  myset contains: 1101 2202 |

### Complexity

Linear in [size](http://www.cplusplus.com/set::size) (destructions).

### Iterator validity

All iterators, pointers and references related to this container are invalidated.

### Data races

The container is modified.  
All contained elements are modified.

### Exception safety

**No-throw guarantee:** this member function never throws exceptions.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::emplace

template <class... Args>

pair<iterator,bool> emplace (Args&&... args);

Construct and insert element

Inserts a new element in the [set](http://www.cplusplus.com/set), if unique. This new element is constructed in place using *args* as the arguments for its construction.  
  
The insertion only takes place if no other element in the container is equivalent to the one being emplaced (elements in a [set](http://www.cplusplus.com/set) container are unique).  
  
If inserted, this effectively increases the container [size](http://www.cplusplus.com/set::size) by one.  
  
Internally, [set](http://www.cplusplus.com/set) containers keep all their elements sorted following the criterion specified by its [comparison object](http://www.cplusplus.com/set::key_comp). The element is always inserted in its respective position following this ordering.  
  
The element is constructed in-place by calling [allocator\_traits::construct](http://www.cplusplus.com/allocator_traits::construct) with *args* forwarded.  
  
A similar member function exists, [insert](http://www.cplusplus.com/set::insert), which either copies or moves existing objects into the container.

### Parameters

args

Arguments forwarded to construct the new element.

### Return value

If the function successfully inserts the element (because no equivalent element existed already in the [set](http://www.cplusplus.com/set)), the function returns a [pair](http://www.cplusplus.com/pair) of an iterator to the newly inserted element and a value of true.  
  
Otherwise, it returns an iterator to the equivalent element within the container and a value of false.  
  
Member type iterator is a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) type that points to an element.  
[pair](http://www.cplusplus.com/pair) is a class template declared in [<utility>](http://www.cplusplus.com/%3Cutility%3E) (see [pair](http://www.cplusplus.com/pair)).

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 | // set::emplace  #include <iostream>  #include <set>  #include <string>  int main ()  {  std::set<std::string> myset;  myset.emplace("foo");  myset.emplace("bar");  auto ret = myset.emplace("foo");  if (!ret.second) std::cout << "foo already exists in myset\n";  return 0;  } |

Output:

|  |
| --- |
| foo already exists in myset |

### Complexity

Logarithmic in the container [size](http://www.cplusplus.com/set::size).

### 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

**Strong guarantee:** if an exception is thrown, there are no changes in the container.  
If [allocator\_traits::construct](http://www.cplusplus.com/allocator_traits::construct) is not supported with the appropriate arguments, it causes *undefined behavior*.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::emplace\_hint

template <class... Args>

iterator emplace\_hint (const\_iterator position, Args&&... args);

Construct and insert element with hint

Inserts a new element in the [set](http://www.cplusplus.com/set), if unique, with a hint on the insertion *position*. This new element is constructed in place using *args* as the arguments for its construction.  
  
The insertion only takes place if no other element in the container is equivalent to the one being emplaced (elements in a [set](http://www.cplusplus.com/set) container are unique).  
  
If inserted, this effectively increases the container [size](http://www.cplusplus.com/set::size) by one.  
  
The value in *position* is used as a hint on the insertion point. The element will nevertheless be inserted at its corresponding position following the order described by its internal [comparison object](http://www.cplusplus.com/set::key_comp), but this hint is used by the function to begin its search for the insertion point, speeding up the process considerably when the actual insertion point is either *position* or close to it.  
  
The element is constructed in-place by calling [allocator\_traits::construct](http://www.cplusplus.com/allocator_traits::construct) with *args* forwarded.

### Parameters

position

Hint for the position where the element can be inserted.  
The function optimizes its insertion time if *position* points to the element that will follow the inserted element (or to the [end](http://www.cplusplus.com/set::end), if it would be the last).  
Notice that this does not force the new element to be inserted at that position within the [set](http://www.cplusplus.com/set) container (the elements in a [set](http://www.cplusplus.com/set) always follow a specific order).  
const\_iterator is a member type, defined as a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) type that points to elements.

args

Arguments forwarded to construct the new element.

### Return value

If the function successfully inserts the element (because no equivalent element existed already in the [set](http://www.cplusplus.com/set)), the function returns an iterator to the newly inserted element.  
  
Otherwise, it returns an iterator to the equivalent element within the container.  
  
Member type iterator is a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) type that points to an element.

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 | // set::emplace\_hint  #include <iostream>  #include <set>  #include <string>  int main ()  {  std::set<std::string> myset;  auto it = myset.cbegin();  myset.emplace\_hint (it,"alpha");  it = myset.emplace\_hint (myset.cend(),"omega");  it = myset.emplace\_hint (it,"epsilon");  it = myset.emplace\_hint (it,"beta");  std::cout << "myset contains:";  for (const std::string& x: myset)  std::cout << ' ' << x;  std::cout << '\n';  return 0;  } |

Output:

|  |
| --- |
| myset contains: alpha beta epsilon omega |

### Complexity

Generally, logarithmic in the container [size](http://www.cplusplus.com/set::size).  
Amortized constant if the insertion point for the element is *position*.

### 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

**Strong guarantee:** if an exception is thrown, there are no changes in the container.  
If [allocator\_traits::construct](http://www.cplusplus.com/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*.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::key\_comp

key\_compare key\_comp() const;

Return comparison object

Returns a copy of the *comparison object* used by the container.  
  
By default, this is a [less](http://www.cplusplus.com/less) object, which returns the same as operator<.  
  
This object determines the order of the elements in the container: it is a function pointer or a function object that takes two arguments of the same type as the container elements, and returns true if the first argument is considered to go before the second in the *strict weak ordering* it defines, and false otherwise.  
  
Two elements of a [set](http://www.cplusplus.com/set) are considered equivalent if key\_comp returns false reflexively (i.e., no matter the order in which the elements are passed as arguments).  
  
In [set](http://www.cplusplus.com/set) containers, the keys to sort the elements are the values themselves, therefore key\_comp and its sibling member function [value\_comp](http://www.cplusplus.com/set::value_comp) are equivalent.

### Parameters

none

### Return value

The comparison object.  
Member type key\_compare is the type of the *comparison object* associated to the container, defined in [set](http://www.cplusplus.com/set) as an alias of its second template parameter (Compare).

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 | // set::key\_comp  #include <iostream>  #include <set>  int main ()  {  std::set<int> myset;  int highest;  std::set<int>::key\_compare mycomp = myset.key\_comp();  for (int i=0; i<=5; i++) myset.insert(i);  std::cout << "myset contains:";  highest=\*myset.rbegin();  std::set<int>::iterator it=myset.begin();  do {  std::cout << ' ' << \*it;  } while ( mycomp(\*(++it),highest) );  std::cout << '\n';  return 0;  } |

Output:

|  |
| --- |
| myset contains: 0 1 2 3 4 |

### Complexity

Constant.

### Iterator validity

No changes.

### Data races

The container is accessed.  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

### Exception safety

**Strong guarantee:** if an exception is thrown, there are no changes in the container.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::value\_comp

value\_compare value\_comp() const;

Return comparison object

Returns a copy of the *comparison object* used by the container.  
  
By default, this is a [less](http://www.cplusplus.com/less) object, which returns the same as operator<.  
  
This object determines the order of the elements in the container: it is a function pointer or a function object that takes two arguments of the same type as the container elements, and returns true if the first argument is considered to go before the second in the *strict weak ordering* it defines, and false otherwise.  
  
Two elements of a [set](http://www.cplusplus.com/set) are considered equivalent if value\_comp returns false reflexively (i.e., no matter the order in which the elements are passed as arguments).  
  
In [set](http://www.cplusplus.com/set) containers, the keys to sort the elements are the values themselves, therefore value\_comp and its sibling member function [key\_comp](http://www.cplusplus.com/key_comp) are equivalent.

### Parameters

none

### Return value

The comparison object.  
Member type value\_compare is the type of the *comparison object* associated to the container, defined in [set](http://www.cplusplus.com/set) as an alias of its second template parameter (Compare).

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 | // set::value\_comp  #include <iostream>  #include <set>  int main ()  {  std::set<int> myset;  std::set<int>::value\_compare mycomp = myset.value\_comp();  for (int i=0; i<=5; i++) myset.insert(i);  std::cout << "myset contains:";  int highest=\*myset.rbegin();  std::set<int>::iterator it=myset.begin();  do {  std::cout << ' ' << \*it;  } while ( mycomp(\*(++it),highest) );  std::cout << '\n';  return 0;  } |

Output:

|  |
| --- |
| myset contains: 0 1 2 3 4 |

### Complexity

Constant.

### Iterator validity

No changes.

### Data races

The container is accessed.  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

### Exception safety

**Strong guarantee:** if an exception is thrown, there are no changes in the container.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::find

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

iterator find (const value\_type& val) const;

Get iterator to element

Searches the container for an element equivalent to *val* and returns an iterator to it if found, otherwise it returns an iterator to [set::end](http://www.cplusplus.com/set::end).  
  
Two elements of a [set](http://www.cplusplus.com/set) are considered equivalent if the container's [comparison object](http://www.cplusplus.com/set::key_comp) returns false reflexively (i.e., no matter the order in which the elements are passed as arguments).

### Parameters

val

Value to be searched for.  
Member type value\_type is the type of the elements in the container, defined in [set](http://www.cplusplus.com/set) as an alias of its first template parameter (T).

### Return value

An iterator to the element, if *val* is found, or [set::end](http://www.cplusplus.com/set::end) otherwise.  
  
Member types iterator and const\_iterator are [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) types pointing to elements.

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 | // set::find  #include <iostream>  #include <set>  int main ()  {  std::set<int> myset;  std::set<int>::iterator it;  // set some initial values:  for (int i=1; i<=5; i++) myset.insert(i\*10); // set: 10 20 30 40 50  it=myset.find(20);  myset.erase (it);  myset.erase (myset.find(40));  std::cout << "myset contains:";  for (it=myset.begin(); it!=myset.end(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  return 0;  } |

Output:

|  |
| --- |
| myset contains: 10 30 50 |

### Complexity

Logarithmic in [size](http://www.cplusplus.com/set::size).

### Iterator validity

No changes.

### Data races

The container is accessed (neither the const nor the non-const versions modify the container).  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

### Exception safety

**Strong guarantee:** if an exception is thrown, there are no changes in the container.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::count

size\_type count (const value\_type& val) const;

Count elements with a specific value

Searches the container for elements equivalent to *val* and returns the number of matches.  
  
Because all elements in a [set](http://www.cplusplus.com/set) container are unique, the function can only return *1* (if the element is found) or zero (otherwise).  
  
Two elements of a [set](http://www.cplusplus.com/set) are considered equivalent if the container's [comparison object](http://www.cplusplus.com/set::key_comp) returns false reflexively (i.e., no matter the order in which the elements are passed as arguments).

### Parameters

val

Value to search for.  
Member type value\_type is the type of the elements in the container, defined in [set](http://www.cplusplus.com/set) as an alias of its first template parameter (T).

### Return value

1 if the container contains an element equivalent to *val*, or zero otherwise.  
  
Member type size\_type is an unsigned integral type.

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 | // set::count  #include <iostream>  #include <set>  int main ()  {  std::set<int> myset;  // set some initial values:  for (int i=1; i<5; ++i) myset.insert(i\*3); // set: 3 6 9 12  for (int i=0; i<10; ++i)  {  std::cout << i;  if (myset.count(i)!=0)  std::cout << " is an element of myset.\n";  else  std::cout << " is not an element of myset.\n";  }  return 0;  } |

Output:

|  |
| --- |
| 0 is not an element of myset.  1 is not an element of myset.  2 is not an element of myset.  3 is an element of myset.  4 is not an element of myset.  5 is not an element of myset.  6 is an element of myset.  7 is not an element of myset.  8 is not an element of myset.  9 is an element of myset. |

### Complexity

Logarithmic in [size](http://www.cplusplus.com/set::size).

### Iterator validity

No changes.

### Data races

The container is accessed.  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

### Exception safety

**Strong guarantee:** if an exception is thrown, there are no changes in the container.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::lower\_bound

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

iterator lower\_bound (const value\_type& val) const;

Return iterator to lower bound

Returns an iterator pointing to the first element in the container which is not considered to go before *val* (i.e., either it is equivalent or goes after).  
  
The function uses its internal [comparison object](http://www.cplusplus.com/set::key_comp) ([key\_comp](http://www.cplusplus.com/set::key_comp)) to determine this, returning an iterator to the first element for which key\_comp(element,val) would return false.  
  
If the [set](http://www.cplusplus.com/set) class is instantiated with the default comparison type ([less](http://www.cplusplus.com/less)), the function returns an iterator to the first element that is not less than *val*.  
  
A similar member function, [upper\_bound](http://www.cplusplus.com/set::upper_bound), has the same behavior as lower\_bound, except in the case that the [set](http://www.cplusplus.com/set) contains an element equivalent to *val*: In this case lower\_bound returns an iterator pointing to that element, whereas [upper\_bound](http://www.cplusplus.com/set::upper_bound) returns an iterator pointing to the next element.

### Parameters

val

Value to compare.  
Member type value\_type is the type of the elements in the container, defined in [set](http://www.cplusplus.com/set) as an alias of its first template parameter (T).

### Return value

An iterator to the the first element in the container which is not considered to go before *val*, or [set::end](http://www.cplusplus.com/set::end) if all elements are considered to go before *val*.  
  
Member types iterator and const\_iterator are [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) types pointing to elements.

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 | // set::lower\_bound/upper\_bound  #include <iostream>  #include <set>  int main ()  {  std::set<int> myset;  std::set<int>::iterator itlow,itup;  for (int i=1; i<10; i++) myset.insert(i\*10); // 10 20 30 40 50 60 70 80 90  itlow=myset.lower\_bound (30); // ^  itup=myset.upper\_bound (60); // ^  myset.erase(itlow,itup); // 10 20 70 80 90  std::cout << "myset contains:";  for (std::set<int>::iterator it=myset.begin(); it!=myset.end(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  return 0;  } |

Notice that lower\_bound(30) returns an iterator to 30, whereas upper\_bound(60) returns an iterator to 70.

|  |
| --- |
| myset contains: 10 20 70 80 90 |

### Complexity

Logarithmic in [size](http://www.cplusplus.com/set::size).

### Iterator validity

No changes.

### Data races

The container is accessed (neither the const nor the non-const versions modify the container).  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

### Exception safety

**Strong guarantee:** if an exception is thrown, there are no changes in the container.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::upper\_bound

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

iterator upper\_bound (const value\_type& val) const;

Return iterator to upper bound

Returns an iterator pointing to the first element in the container which is considered to go after *val*.  
  
The function uses its internal [comparison object](http://www.cplusplus.com/set::key_comp) ([key\_comp](http://www.cplusplus.com/set::key_comp)) to determine this, returning an iterator to the first element for which key\_comp(val,element) would return true.  
  
If the [set](http://www.cplusplus.com/set) class is instantiated with the default comparison type ([less](http://www.cplusplus.com/less)), the function returns an iterator to the first element that is greater than *val*.  
  
A similar member function, [lower\_bound](http://www.cplusplus.com/set::lower_bound), has the same behavior as upper\_bound, except in the case that the [set](http://www.cplusplus.com/set) contains an element equivalent to *val*: In this case [lower\_bound](http://www.cplusplus.com/set::lower_bound) returns an iterator pointing to that element, whereas upper\_bound returns an iterator pointing to the next element.

### Parameters

val

Value to compare.  
Member type value\_type is the type of the elements in the container, defined in [set](http://www.cplusplus.com/set) as an alias of its first template parameter (T).

### Return value

An iterator to the the first element in the container which is considered to go after *val*, or [set::end](http://www.cplusplus.com/set::end) if no elements are considered to go after *val*.  
  
Member types iterator and const\_iterator are [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) types pointing to elements.

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 | // set::lower\_bound/upper\_bound  #include <iostream>  #include <set>  int main ()  {  std::set<int> myset;  std::set<int>::iterator itlow,itup;  for (int i=1; i<10; i++) myset.insert(i\*10); // 10 20 30 40 50 60 70 80 90  itlow=myset.lower\_bound (30); // ^  itup=myset.upper\_bound (60); // ^  myset.erase(itlow,itup); // 10 20 70 80 90  std::cout << "myset contains:";  for (std::set<int>::iterator it=myset.begin(); it!=myset.end(); ++it)  std::cout << ' ' << \*it;  std::cout << '\n';  return 0;  } |

Notice that lower\_bound(30) returns an iterator to 30, whereas upper\_bound(60) returns an iterator to 70.

|  |
| --- |
| myset contains: 10 20 70 80 90 |

### Complexity

Logarithmic in [size](http://www.cplusplus.com/set::size).

### Iterator validity

No changes.

### Data races

The container is accessed (neither the const nor the non-const versions modify the container).  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

### Exception safety

**Strong guarantee:** if an exception is thrown, there are no changes in the container.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::equal\_range

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

pair<iterator,iterator> equal\_range (const value\_type& val) const;

Get range of equal elements

Returns the bounds of a range that includes all the elements in the container that are equivalent to *val*.  
  
Because all elements in a [set](http://www.cplusplus.com/set) container are unique, the range returned will contain a single element at most.  
  
If no matches are found, the range returned has a length of zero, with both iterators pointing to the first element that is considered to go after *val* according to the container's [internal comparison object](http://www.cplusplus.com/set::key_comp) ([key\_comp](http://www.cplusplus.com/set::key_comp)).  
  
Two elements of a [set](http://www.cplusplus.com/set) are considered equivalent if the container's [comparison object](http://www.cplusplus.com/set::key_comp) returns false reflexively (i.e., no matter the order in which the elements are passed as arguments).

### Parameters

val

Value to search for.  
Member type value\_type is the type of the elements in the container, defined in [set](http://www.cplusplus.com/set) as an alias of its first template parameter (T).

### Return value

The function returns a [pair](http://www.cplusplus.com/pair), whose member pair::first is the lower bound of the range (the same as [lower\_bound](http://www.cplusplus.com/set::lower_bound)), and pair::second is the upper bound (the same as [upper\_bound](http://www.cplusplus.com/set::upper_bound)).  
  
Member types iterator and const\_iterator are [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) types pointing to elements.

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 | // set::equal\_elements  #include <iostream>  #include <set>  int main ()  {  std::set<int> myset;  for (int i=1; i<=5; i++) myset.insert(i\*10); // myset: 10 20 30 40 50  std::pair<std::set<int>::const\_iterator,std::set<int>::const\_iterator> ret;  ret = myset.equal\_range(30);  std::cout << "the lower bound points to: " << \*ret.first << '\n';  std::cout << "the upper bound points to: " << \*ret.second << '\n';  return 0;  } |

|  |
| --- |
| the lower bound points to: 30  the upper bound points to: 40 |

### Complexity

Logarithmic in [size](http://www.cplusplus.com/set::size).

### Iterator validity

No changes.

### Data races

The container is accessed (neither the const nor the non-const versions modify the container).  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

### Exception safety

**Strong guarantee:** if an exception is thrown, there are no changes in the container.

public member function

<set>

# std::[set](http://www.cplusplus.com/reference/set/set/)::get\_allocator

* [C++98](javascript:switch1.select(1))
* [C++11](javascript:switch1.select(2))

allocator\_type get\_allocator() const;

Get allocator

Returns a copy of the allocator object associated with the [set](http://www.cplusplus.com/set).

### Parameters

none

### Return Value

The allocator.  
  
Member type allocator\_type is the type of the allocator used by the container, defined in [set](http://www.cplusplus.com/set) as an alias of its third template parameter (Alloc).

### Example

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 | // set::get\_allocator  #include <iostream>  #include <set>  int main ()  {  std::set<int> myset;  int \* p;  unsigned int i;  // allocate an array of 5 elements using myset's allocator:  p=myset.get\_allocator().allocate(5);  // assign some values to array  for (i=0; i<5; i++) p[i]=(i+1)\*10;  std::cout << "The allocated array contains:";  for (i=0; i<5; i++) std::cout << ' ' << p[i];  std::cout << '\n';  myset.get\_allocator().deallocate(p,5);  return 0;  } |

The example shows an elaborate way to allocate memory for an array of ints using the same allocator used by the container.  
Output:

|  |
| --- |
| The allocated array contains: 10 20 30 40 50 |

### Complexity

Constant.

### Iterator validity

No changes.

### Data races

The container is accessed.  
Concurrently accessing the elements of a [set](http://www.cplusplus.com/set) is safe.

### Exception safety

**No-throw guarantee:** this member function never throws exceptions.  
Copying any instantiation of the [default allocator](http://www.cplusplus.com/allocator) is also guaranteed to never throw.