# std::set

Defined in header <set>

```
template<
    class Key,
    class Compare = std::less<Key>,
    class Allocator = std::allocator<Key>
> class set;
```
(1)

```
namespace pmr {
    template<
        class Key,
        class Compare = std::less<Key>
    > using set = std::set<Key, Compare, std::pmr::polymorphic_allocator<Key>>;
}
```
(2) (since C++17)

`std::set` is an associative container that contains a sorted set of unique objects of type Key. Sorting is done using the key comparison function *Compare*. Search, removal, and insertion operations have logarithmic complexity. Sets are usually implemented as Red–black trees .

Everywhere the standard library uses the *Compare* requirements, uniqueness is determined by using the equivalence relation. In imprecise terms, two objects `a` and `b` are considered equivalent if neither compares less than the other: `!comp(a, b) && !comp(b, a)`.

`std::set` meets the requirements of *Container*, *AllocatorAwareContainer*, *AssociativeContainer* and *ReversibleContainer*.

All member functions of `std::set` are `constexpr`: it is possible to create and use `std::set` objects in the evaluation of a constant expression. (since C++26)

However, `std::set` objects generally cannot be `constexpr`, because any dynamically allocated storage must be released in the same evaluation of constant expression.

## Template parameters

This section is incomplete
Reason: Add descriptions of the template parameters.

## Member types

| Type | Definition |
|---|---|
| `key_type` | Key |
| `value_type` | Key |
| `size_type` | Unsigned integer type (usually `std::size_t`) |
| `difference_type` | Signed integer type (usually `std::ptrdiff_t`) |
| `key_compare` | Compare |
| `value_compare` | Compare |
| `allocator_type` | Allocator |
| `reference` | `value_type&` |
| `const_reference` | `const value_type&` |
| `pointer` | `Allocator::pointer` (until C++11)<br>`std::allocator_traits<Allocator>::pointer` (since C++11) |
| `const_pointer` | `Allocator::const_pointer` (until C++11)<br>`std::allocator_traits<Allocator>::const_pointer` (since C++11) |
| `iterator` | Constant *LegacyBidirectionalIterator* and *ConstexprIterator* (since C++26) to `value_type` |
| `const_iterator` | *LegacyBidirectionalIterator* and *ConstexprIterator* (since C++26) to `const value_type` |
| `reverse_iterator` | `std::reverse_iterator<iterator>` |
| `const_reverse_iterator` | `std::reverse_iterator<const_iterator>` |
| `node_type` (since C++17) | a specialization of node handle representing a container node |
| `insert_return_type` (since C++17) | type describing the result of inserting a node_type, a specialization of |

```
template<class Iter, class NodeType>
struct /*unspecified*/
{
    Iter     position;
    bool     inserted;
    NodeType node;
};
```

instantiated with template arguments `iterator` and `node_type`.

## Member functions

| | |
|---|---|
| (constructor) | constructs the `set`<br>(public member function) |
| (destructor) | destructs the `set`<br>(public member function) |
| **`operator=`** | assigns values to the container<br>(public member function) |
| **`get_allocator`** | returns the associated allocator<br>(public member function) |

#### Iterators

| | |
|---|---|
| **`begin`**<br>**`cbegin`** (C++11) | returns an iterator to the beginning<br>(public member function) |
| **`end`**<br>**`cend`** (C++11) | returns an iterator to the end<br>(public member function) |
| **`rbegin`**<br>**`crbegin`** (C++11) | returns a reverse iterator to the beginning<br>(public member function) |
| **`rend`**<br>**`crend`** (C++11) | returns a reverse iterator to the end<br>(public member function) |

#### Capacity

| | |
|---|---|
| **`empty`** | checks whether the container is empty<br>(public member function) |
| **`size`** | returns the number of elements<br>(public member function) |
| **`max_size`** | returns the maximum possible number of elements<br>(public member function) |

#### Modifiers

| | |
|---|---|
| **`clear`** | clears the contents<br>(public member function) |
| **`insert`** | inserts elements or nodes(since C++17)<br>(public member function) |
| **`insert_range`** (C++23) | inserts a range of elements<br>(public member function) |
| **`emplace`** (C++11) | constructs element in-place<br>(public member function) |
| **`emplace_hint`** (C++11) | constructs elements in-place using a hint<br>(public member function) |
| **`erase`** | erases elements<br>(public member function) |
| **`swap`** | swaps the contents<br>(public member function) |
| **`extract`** (C++17) | extracts nodes from the container<br>(public member function) |
| **`merge`** (C++17) | splices nodes from another container<br>(public member function) |

#### Lookup

| | |
|---|---|
| **`count`** | returns the number of elements matching specific key<br>(public member function) |
| **`find`** | finds element with specific key<br>(public member function) |
| **`contains`** (C++20) | checks if the container contains element with specific key<br>(public member function) |

| | |
|---|---|
| **equal_range** | returns range of elements matching a specific key<br>(public member function) |
| **lower_bound** | returns an iterator to the first element *not less* than the given key<br>(public member function) |
| **upper_bound** | returns an iterator to the first element *greater* than the given key<br>(public member function) |

**Observers**

| | |
|---|---|
| **key_comp** | returns the function that compares keys<br>(public member function) |
| **value_comp** | returns the function that compares keys in objects of type `value_type`<br>(public member function) |

## Non-member functions

| | |
|---|---|
| **operator==**<br>**operator!=** (removed in C++20)<br>**operator<** (removed in C++20)<br>**operator<=** (removed in C++20)<br>**operator>** (removed in C++20)<br>**operator>=** (removed in C++20)<br>**operator<=>** (C++20) | lexicographically compares the values of two sets<br>(function template) |
| **std::swap**(std::set) | specializes the `std::swap` algorithm<br>(function template) |
| **erase_if**(std::set) (C++20) | erases all elements satisfying specific criteria<br>(function template) |

**Deduction guides** (since C++17)

## Notes

The member types `iterator` and `const_iterator` may be aliases to the same type. This means defining a pair of function overloads using the two types as parameter types may violate the One Definition Rule. Since `iterator` is convertible to `const_iterator`, a single function with a `const_iterator` as parameter type will work instead.

| Feature-test macro | Value | Std | Feature |
|---|---|---|---|
| `__cpp_lib_containers_ranges` | 202202L | (C++23) | Ranges construction and insertion for containers |
| `__cpp_lib_constexpr_set` | 202502L | (C++26) | `constexpr` `std::set` |

## Example

Run this code

```
#include <algorithm>
#include <iomanip>
#include <iostream>
#include <iterator>
#include <set>
#include <string_view>

template<typename T>
std::ostream& operator<<(std::ostream& out, const std::set<T>& set)
{
    if (set.empty())
        return out << "{}";
    out << "{ " << *set.begin();
    std::for_each(std::next(set.begin()), set.end(), [&out](const T& element)
    {
        out << ", " << element;
    });
    return out << " }";
}

int main()
{
    std::set<int> set{1, 5, 3};
    std::cout << set << '\n';
```

```
    set.insert(2);
    std::cout << set << '\n';

    set.erase(1);
    std::cout << set << "\n\n";

    std::set<int> keys{3, 4};
    for (int key : keys)
    {
        if (set.contains(key))
            std::cout << set << " does contain " << key << '\n';
        else
            std::cout << set << " doesn't contain " << key << '\n';
    }
    std::cout << '\n';

    std::string_view word = "element";
    std::set<char> characters(word.begin(), word.end());
    std::cout << "There are " << characters.size() << " unique characters in "
              << std::quoted(word) << ":\n" << characters << '\n';
}
```

Output:

```
{ 1, 3, 5 }
{ 1, 2, 3, 5 }
{ 2, 3, 5 }

{ 2, 3, 5 } does contain 3
{ 2, 3, 5 } doesn't contain 4

There are 5 unique characters in "element":
{ e, l, m, n, t }
```

## Defect reports

The following behavior-changing defect reports were applied retroactively to previously published C++ standards.

| DR | Applied to | Behavior as published | Correct behavior |
|---|---|---|---|
| LWG 103 (https://cplusplus.github.io/LWG/issue103) | C++98 | iterator allows modification of keys | iterator made constant |
| LWG 230 (https://cplusplus.github.io/LWG/issue230) | C++98 | Key was not required to be *CopyConstructible*<br>(a key of type Key might not be able to be constructed) | Key is also required to be *CopyConstructible* |

## See also

| | |
|---|---|
| **multiset** | collection of keys, sorted by keys<br>(class template) |
| **unordered_set** (C++11) | collection of unique keys, hashed by keys<br>(class template) |
| **flat_set** (C++23) | adapts a container to provide a collection of unique keys, sorted by keys<br>(class template) |