std::flat_map

Defined in header `<flat_map>`

```
template<
    class Key,
    class T,
    class Compare = std::less<Key>,
    class KeyContainer = std::vector<Key>,
    class MappedContainer = std::vector<T>
> class flat_map;
```
(since C++23)

The flat map is a container adaptor that gives the functionality of an associative container that contains key-value pairs with unique keys. Keys are sorted by using the comparison function `Compare`.

The class template `flat_map` acts as a wrapper to the two underlying containers, passed as objects of type `KeyContainer` and `MappedContainer` respectively. The first container is sorted, and for each key its corresponding value is in the second container at the same index (offset). The number of elements in both containers is the same.

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

`std::flat_map` meets the requirements of *Container*, *ReversibleContainer*, optional container requirements, and all requirements of *AssociativeContainer* (including logarithmic search complexity), except that:

- requirements related to nodes are not applicable,
- iterator invalidation requirements differ,
- the complexity of insertion and erasure operations is linear.

A flat map supports most *AssociativeContainer*'s operations that use unique keys.

All member functions of `std::flat_map` are `constexpr`: it is possible to create and use `std::flat_map` objects in the evaluation of a constant expression.

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

(since C++26)

Iterator invalidation

This section is incomplete

Template parameters

Key	-	The type of the keys. The program is ill-formed if Key is not the same type as `KeyContainer::value_type`.
T	-	The type of mapped values. The program is ill-formed if T is not the same type as `MappedContainer::value_type`.
`Compare`	-	A *Compare* type providing a strict weak ordering.

`KeyContainer` **`MappedContainer`**	-	The types of the underlying *SequenceContainer* to store keys and mapped values correspondingly. The iterators of such containers should satisfy *LegacyRandomAccessIterator* or model `random_access_iterator`. Invocations of their member functions `size` and `max_size` should not exit via an exception. The standard containers `std::vector` and `std::deque` satisfy these requirements.

Member types

Type	**Definition**
`key_container_type`	`KeyContainer`
`mapped_container_type`	`MappedContainer`
`key_type`	`Key`
`mapped_type`	`T`
`value_type`	`std::pair<key_type, mapped_type>`
`key_compare`	`Compare`
`reference`	`std::pair<const key_type&, mapped_type&>`
`const_reference`	`std::pair<const key_type&, const mapped_type&>`
`size_type`	`std::size_t`
`difference_type`	`std::ptrdiff_t`
`iterator`	implementation-defined *LegacyInputIterator*, *ConstexprIterator*(since C++26) and `random_access_iterator` to `value_type`
`const_iterator`	implementation-defined *LegacyInputIterator*, *ConstexprIterator*(since C++26) and `random_access_iterator` to `const value_type`
`reverse_iterator`	`std::reverse_iterator<iterator>`
`const_reverse_iterator`	`std::reverse_iterator<const_iterator>`

`containers`	type describing the underlying containers
	```
	struct containers
	{
	    key_container_type keys;
	    mapped_container_type values;
	};
	```

Member classes

`value_compare`	compares objects of type `value_type` (class)

Member objects

Member	**Description**
`containers` *`c`* (private)	the adapted containers (exposition-only member object*)
`key_compare` *`compare`* (private)	the comparison function object (exposition-only member object*)

Member functions

(constructor)	constructs the `flat_map` (public member function)
(destructor) (implicitly declared)	destroys every element of the container adaptor (public member function)
`operator=`	assigns values to the container adaptor (public member function)

Element access

`at`	access specified element with bounds checking (public member function)
`operator[]`	access or insert specified element (public member function)

Iterators

`begin` **`cbegin`**	returns an iterator to the beginning (public member function)
`end` **`cend`**	returns an iterator to the end (public member function)
`rbegin` **`crbegin`**	returns a reverse iterator to the beginning (public member function)
`rend` **`crend`**	returns a reverse iterator to the end (public member function)

Capacity

`empty`	checks whether the container adaptor is empty (public member function)
`size`	returns the number of elements (public member function)
`max_size`	returns the maximum possible number of elements (public member function)

Modifiers

`emplace`	constructs element in-place (public member function)
`emplace_hint`	constructs elements in-place using a hint (public member function)
`try_emplace`	inserts in-place if the key does not exist, does nothing if the key exists (public member function)
`insert`	inserts elements (public member function)
`insert_range`	inserts a range of elements (public member function)
`insert_or_assign`	inserts an element or assigns to the current element if the key already exists (public member function)
`extract`	extracts the underlying containers (public member function)
`replace`	replaces the underlying containers (public member function)

erase	erases elements (public member function)
swap	swaps the contents (public member function)
clear	clears the contents (public member function)

Lookup

find	finds element with specific key (public member function)
count	returns the number of elements matching specific key (public member function)
contains	checks if the container contains element with specific key (public member function)
lower_bound	returns an iterator to the first element *not less* than the given key (public member function)
upper_bound	returns an iterator to the first element *greater* than the given key (public member function)
equal_range	returns range of elements matching a specific key (public member function)

Observers

key_comp	returns the function that compares keys (public member function)
value_comp	returns the function that compares keys in objects of type `value_type` (public member function)
keys	direct access to the underlying keys container (public member function)
values	direct access to the underlying values container (public member function)

Non-member functions

operator== **operator<=>** (C++23)	lexicographically compares the values of two `flat_map`s (function template)
std::swap(std::flat_map) (C++23)	specializes the `std::swap` algorithm (function template)
erase_if(std::flat_map) (C++23)	erases all elements satisfying specific criteria (function template)

Helper classes

std::uses_allocator<std::flat_map> (C++23)	specializes the `std::uses_allocator` type trait (class template specialization)

Tags

sorted_unique **sorted_unique_t** (C++23)	indicates that elements of a range are sorted and unique (tag)

Deduction guides

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_flat_map`	`202207L`	(C++23)	`std::flat_map` and `std::flat_multimap`
`__cpp_lib_constexpr_containers`	`202502L`	(C++26)	`constexpr std::flat_map`

Example

This section is incomplete
Reason: no example

See also

`flat_multimap` (C++23)	adapts two containers to provide a collection of key-value pairs, sorted by keys (class template)
`map`	collection of key-value pairs, sorted by keys, keys are unique (class template)
`unordered_map` (C++11)	collection of key-value pairs, hashed by keys, keys are unique (class template)

Retrieved from "https://en.cppreference.com/mwiki/index.php?title=cpp/container/flat_map&oldid=180917"