



map<Key, Data, Compare, Alloc>

Containers

Category: containers

Type

Component type: type

Description

Map is a [Sorted Associative Container](#) that associates objects of type Key with objects of type Data. Map is a [Pair Associative Container](#), meaning that its value type is `pair<const Key, Data>`. It is also a [Unique Associative Container](#), meaning that no two elements have the same key.

Map has the important property that inserting a new element into a map does not invalidate iterators that point to existing elements. Erasing an element from a map also does not invalidate any iterators, except, of course, for iterators that actually point to the element that is being erased.

Example

```
struct ltstr
{
    bool operator()(const char* s1, const char* s2) const
    {
        return strcmp(s1, s2) < 0;
    }
};

int main()
{
    map<const char*, int, ltstr> months;

    months["january"] = 31;
    months["february"] = 28;
    months["march"] = 31;
    months["april"] = 30;
    months["may"] = 31;
    months["june"] = 30;
    months["july"] = 31;
    months["august"] = 31;
    months["september"] = 30;
    months["october"] = 31;
    months["november"] = 30;
    months["december"] = 31;

    cout << "june -> " << months["june"] << endl;
    map<const char*, int, ltstr>::iterator cur = months.find("june");
    map<const char*, int, ltstr>::iterator prev = cur;
    map<const char*, int, ltstr>::iterator next = cur;
    ++next;
    --prev;
    cout << "Previous (in alphabetical order) is " << (*prev).first << endl;
    cout << "Next (in alphabetical order) is " << (*next).first << endl;
}
```

Definition

Defined in the standard header [map](#), and in the nonstandard backward-compatibility header [map.h](#).

Template parameters

Parameter	Description	Default
Key	The map's key type. This is also defined as <code>map::key_type</code> .	
Data	The map's data type. This is also defined as <code>map::data_type</code> .	
Compare	The key comparison function, a Strict Weak Ordering whose argument type is <code>key_type</code> ; it returns <code>true</code> if its first argument is less than its second argument, and <code>false</code> otherwise. This is also defined as <code>map::key_compare</code> .	less<Key>
Alloc	The map's allocator, used for all internal memory management.	alloc

Model of

[Unique Sorted Associative Container](#), [Pair Associative Container](#)

Type requirements

- Data is [Assignable](#).
- Compare is a [Strict Weak Ordering](#) whose argument type is Key.
- Alloc is an [Allocator](#).

Public base classes

None.

Members

Member	Where defined	Description
<code>key_type</code>	Associative Container	The map's key type, Key.
<code>data_type</code>	Pair Associative Container	The type of object associated with the keys.
<code>value_type</code>	Pair Associative Container	The type of object, <code>pair<const key_type, data_type></code> , stored in the map.
<code>key_compare</code>	Sorted Associative Container	Function object that compares two keys for ordering.
<code>value_compare</code>	Sorted Associative Container	Function object that compares two values for ordering.
<code>pointer</code>	Container	Pointer to T.
<code>reference</code>	Container	Reference to T
<code>const_reference</code>	Container	Const reference to T
<code>size_type</code>	Container	An unsigned integral type.
<code>difference_type</code>	Container	A signed integral type.
<code>iterator</code>	Container	Iterator used to iterate through a map. [1]
<code>const_iterator</code>	Container	Const iterator used to iterate through a map.
<code>reverse_iterator</code>	Reversible Container	Iterator used to iterate backwards through a map. [1]
<code>const_reverse_iterator</code>	Reversible Container	Const iterator used to iterate backwards through a map.

<code>iterator begin()</code>	Container	Returns an <code>iterator</code> pointing to the beginning of the map.
<code>iterator end()</code>	Container	Returns an <code>iterator</code> pointing to the end of the map.
<code>const_iterator begin() const</code>	Container	Returns a <code>const_iterator</code> pointing to the beginning of the map.
<code>const_iterator end() const</code>	Container	Returns a <code>const_iterator</code> pointing to the end of the map.
<code>reverse_iterator rbegin()</code>	Reversible Container	Returns a <code>reverse_iterator</code> pointing to the beginning of the reversed map.
<code>reverse_iterator rend()</code>	Reversible Container	Returns a <code>reverse_iterator</code> pointing to the end of the reversed map.
<code>const_reverse_iterator rbegin() const</code>	Reversible Container	Returns a <code>const_reverse_iterator</code> pointing to the beginning of the reversed map.
<code>const_reverse_iterator rend() const</code>	Reversible Container	Returns a <code>const_reverse_iterator</code> pointing to the end of the reversed map.
<code>size_type size() const</code>	Container	Returns the size of the map.
<code>size_type max_size() const</code>	Container	Returns the largest possible size of the map.
<code>bool empty() const</code>	Container	true if the map's size is 0.
<code>key_compare key_comp() const</code>	Sorted Associative Container	Returns the <code>key_compare</code> object used by the map.
<code>value_compare value_comp() const</code>	Sorted Associative Container	Returns the <code>value_compare</code> object used by the map.
<code>map()</code>	Container	Creates an empty map.
<code>map(const key_compare& comp)</code>	Sorted Associative Container	Creates an empty map, using <code>comp</code> as the <code>key_compare</code> object.
<code>template <class InputIterator> map(InputIterator f, InputIterator l) [2]</code>	Unique Sorted Associative Container	Creates a map with a copy of a range.
<code>template <class InputIterator> map(InputIterator f, InputIterator l, const key_compare& comp) [2]</code>	Unique Sorted Associative Container	Creates a map with a copy of a range, using <code>comp</code> as the <code>key_compare</code> object.
<code>map(const map&)</code>	Container	The copy constructor.
<code>map& operator=(const map&)</code>	Container	The assignment operator
<code>void swap(map&)</code>	Container	Swaps the contents of two maps.
<code>pair<iterator, bool> insert(const value_type& x)</code>	Unique Associative Container	Inserts <code>x</code> into the map.
<code>iterator insert(iterator pos, const value_type& x)</code>	Unique Sorted Associative Container	Inserts <code>x</code> into the map, using <code>pos</code> as a hint to where it will be inserted.

template <class InputIterator > void insert(InputIterator, InputIterator) [2]	Unique Sorted Associative Container	Inserts a range into the map.
void erase(iterator pos)	Associative Container	Erases the element pointed to by pos.
size_type erase(const key_type& k)	Associative Container	Erases the element whose key is k.
void erase(iterator first, iterator last)	Associative Container	Erases all elements in a range.
void clear()	Associative Container	Erases all of the elements.
iterator find(const key_type& k)	Associative Container	Finds an element whose key is k.
const_iterator find(const key_type& k) const	Associative Container	Finds an element whose key is k.
size_type count(const key_type& k)	Unique Associative Container	Counts the number of elements whose key is k.
iterator lower_bound(const key_type& k)	Sorted Associative Container	Finds the first element whose key is not less than k.
const_iterator lower_bound(const key_type& k) const	Sorted Associative Container	Finds the first element whose key is not less than k.
iterator upper_bound(const key_type& k)	Sorted Associative Container	Finds the first element whose key greater than k.
const_iterator upper_bound(const key_type& k) const	Sorted Associative Container	Finds the first element whose key greater than k.
pair<iterator, iterator> equal_range(const key_type& k)	Sorted Associative Container	Finds a range containing all elements whose key is k.
pair<const_iterator, const_iterator> equal_range(const key_type& k) const	Sorted Associative Container	Finds a range containing all elements whose key is k.
data_type& operator[](const key_type& k) [3]	map	See below.
bool operator==(const map&, const map&)	Forward Container	Tests two maps for equality. This is a global function, not a member function.
bool operator<(const map&, const map&)	Forward Container	Lexicographical comparison. This is a global function, not a member function.

New members

These members are not defined in the [Unique Sorted Associative Container](#) and [Pair Associative Container](#) requirements, but are unique to map:

Member function	Description
data_type& operator[](const key_type& k) [3]	Returns a reference to the object that is associated with a particular key. If the map does not already contain such an object, operator[] inserts the default object data_type(). [3]

Notes

[1] `map::iterator` is not a mutable iterator, because `map::value_type` is not [Assignable](#). That is, if `i` is of type `map::iterator` and `p` is of type `map::value_type`, then `*i = p` is not a valid expression. However, `map::iterator` isn't a constant iterator either, because it can be used to modify the object that it points to. Using the same notation as above, `(*i).second = p.second` is a valid expression. The same point applies to `map::reverse_iterator`.

[2] This member function relies on *member template* functions, which at present (early 1998) are not supported by all compilers. If your compiler supports member templates, you can call this function with any type of [input iterator](#). If your compiler does not yet support member templates, though, then the arguments must either be of type `const value_type*` or of type `map::const_iterator`.

[3] Since `operator[]` might insert a new element into the map, it can't possibly be a `const` member function. Note that the definition of `operator[]` is extremely simple: `m[k]` is equivalent to `((*(m.insert(value_type(k, data_type()))).first)).second`. Strictly speaking, this member function is unnecessary: it exists only for convenience.

See also

[Associative Container](#), [Sorted Associative Container](#), [Pair Associative Container](#), [Unique Sorted Associative Container](#), [set multiset](#), [multimap](#), [hash_set](#), [hash_map](#), [hash_multiset](#), [hash_multimap](#),

[STL Main Page](#)

[Contact Us](#) | [Site Map](#) | [Trademarks](#) | [Privacy](#) | Using this site means you accept its [Terms of Use](#)
Copyright © 2009 - 2011 Silicon Graphics International. All rights reserved.