

Search:

[Reference](#) [Containers](#)

Not logged in
[register](#) [log in](#)

C++

Information
Tutorials
Reference
Articles
Forum

Reference

C library:

Containers:

<array> C++11
<deque>
<forward_list> C++11
<list>
<map>
<queue>
<set>
<stack>
<unordered_map> C++11
<unordered_set> C++11
<vector>

Input/Output:
Multi-threading:
Other:

Upstart® Lending

Up To \$35k w/ APR Starting at 4.7%.
640+ Credit. Check Your Rate Now!

library

Containers

Standard Containers

A container is a holder object that stores a collection of other objects (its elements). They are implemented as class templates, which allows a great flexibility in the types supported as elements.

The container manages the storage space for its elements and provides member functions to access them, either directly or through iterators (reference objects with similar properties to pointers).

Containers replicate structures very commonly used in programming: dynamic arrays ([vector](#)), queues ([queue](#)), stacks ([stack](#)), heaps ([priority_queue](#)), linked lists ([list](#)), trees ([set](#)), associative arrays ([map](#))...

Many containers have several member functions in common, and share functionalities. The decision of which type of container to use for a specific need does not generally depend only on the functionality offered by the container, but also on the efficiency of some of its members (complexity). This is especially true for sequence containers, which offer different trade-offs in complexity between inserting/removing elements and accessing them.

[stack](#), [queue](#) and [priority_queue](#) are implemented as *container adaptors*. Container adaptors are not full container classes, but classes that provide a specific interface relying on an object of one of the container classes (such as [deque](#) or [list](#)) to handle the elements. The underlying container is encapsulated in such a way that its elements are accessed by the members of the *container adaptor* independently of the underlying *container* class used.

Container class templates

Sequence containers:

array <small>C++11</small>	Array class (class template)
vector	Vector (class template)
deque	Double ended queue (class template)
forward_list <small>C++11</small>	Forward list (class template)
list	List (class template)

Container adaptors:

stack	LIFO stack (class template)
queue	FIFO queue (class template)
priority_queue	Priority queue (class template)

Associative containers:

set	Set (class template)
multiset	Multiple-key set (class template)
map	Map (class template)
multimap	Multiple-key map (class template)

Unordered associative containers:

unordered_set <small>C++11</small>	Unordered Set (class template)
unordered_multiset <small>C++11</small>	Unordered Multiset (class template)
unordered_map <small>C++11</small>	Unordered Map (class template)
unordered_multimap <small>C++11</small>	Unordered Multimap (class template)

Other:

Two class templates share certain properties with containers, and are sometimes classified with them: [bitset](#) and [valarray](#).

Member map

This is a comparison chart with the different member functions present on each of the different containers:

Legend:
C++98 Available since C++98
C++11 New in C++11


Sequence containers

Headers	<array>	<vector>	<deque>	<forward_list>	<list>
Members	array	vector	deque	forward_list	list
	constructor	implicit	vector	deque	forward_list
	destructor	implicit	~vector	~deque	~forward_list
	operator=	implicit	operator=	operator=	operator=
iterators	begin	begin	begin	begin	before_begin
	end	end	end	end	end
	rbegin	rbegin	rbegin		rbegin
	rend	rend	rend		rend
const iterators	cbegin	cbegin	cbegin	cbegin	cbefore_begin
	cend	cend	cend	cend	cend
	crbegin	crbegin	crbegin		crbegin
	crend	crend	crend		crend
	size	size	size		size
	max_size	max_size	max_size	max_size	max_size

capacity	empty	empty	empty	empty	empty	empty
	resize		resize	resize	resize	resize
	shrink_to_fit		shrink_to_fit	shrink_to_fit		
	capacity		capacity			
element access	reserve		reserve			
	front	front	front	front	front	front
	back	back	back	back		back
	operator[]	operator[]	operator[]	operator[]		
modifiers	at	at	at	at		
	assign		assign	assign	assign	assign
	emplace		emplace	emplace	emplace_after	emplace
	insert		insert	insert	insert_after	insert
	erase		erase	erase	erase_after	erase
	emplace_back		emplace_back	emplace_back		emplace_back
	push_back		push_back	push_back		push_back
	pop_back		pop_back	pop_back		pop_back
	emplace_front			emplace_front	emplace_front	emplace_front
	push_front			push_front	push_front	push_front
	pop_front			pop_front	pop_front	pop_front
	clear		clear	clear	clear	clear
	swap	swap	swap	swap	swap	swap
	splice				splice_after	splice
list operations	remove				remove	remove
	remove_if				remove_if	remove_if
	unique				unique	unique
	merge				merge	merge
	sort				sort	sort
	reverse				reverse	reverse
observers	get_allocator		get_allocator	get_allocator	get_allocator	get_allocator
	data	data	data			

Associative containers

Headers		<set>		<map>		<unordered_set>		
Members		set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map
	constructor	set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map
	destructor	~set	~multiset	~map	~multimap	~unordered_set	~unordered_multiset	~unordered_map
	assignment	operator=	operator=	operator=	operator=	operator=	operator=	operator=
iterators	begin	begin	begin	begin	begin	begin	begin	begin
	end	end	end	end	end	end	end	end
	rbegin	rbegin	rbegin	rbegin	rbegin			
	rend	rend	rend	rend	rend			
const iterators	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin
	cend	cend	cend	cend	cend	cend	cend	cend
	crbegin	crbegin	crbegin	crbegin	crbegin			
	crend	crend	crend	crend	crend			
capacity	size	size	size	size	size	size	size	size
	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size
	empty	empty	empty	empty	empty	empty	empty	empty
	reserve					reserve	reserve	reserve
element access	at			at				at
	operator[]			operator[]				operator[]
modifiers	emplace	emplace	emplace	emplace	emplace	emplace	emplace	emplace
	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint
	insert	insert	insert	insert	insert	insert	insert	insert
	erase	erase	erase	erase	erase	erase	erase	erase
	clear	clear	clear	clear	clear	clear	clear	clear
	swap	swap	swap	swap	swap	swap	swap	swap
operations	count	count	count	count	count	count	count	count
	find	find	find	find	find	find	find	find
	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range
	lower_bound	lower_bound	lower_bound	lower_bound	lower_bound			
observers	upper_bound	upper_bound	upper_bound	upper_bound	upper_bound			
	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator
	key_comp	key_comp	key_comp	key_comp	key_comp			
	value_comp	value_comp	value_comp	value_comp	value_comp			
buckets	key_eq					key_eq	key_eq	key_eq
	hash_function					hash_function	hash_function	hash_function
	bucket					bucket	bucket	bucket
	bucket_count					bucket_count	bucket_count	bucket_count
hash policy	bucket_size					bucket_size	bucket_size	bucket_size
	max_bucket_count					max_bucket_count	max_bucket_count	max_bucket_count
	rehash					rehash	rehash	rehash
	load_factor					load_factor	load_factor	load_factor
	max_load_factor					max_load_factor	max_load_factor	max_load_factor



Because C++

