Containers library

The Containers library is a generic collection of class templates and algorithms that allow programmers to easily implement common data structures like queues, lists and stacks. There are three classes of containers -- sequence containers, associative containers, and unordered associative containers -- each of which is designed to support a different set of operations.

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

Most containers have at least several member functions in common, and share functionalities. Which container is the best for the particular application depends not only on the offered functionality, but also on its efficiency for different workloads.

Sequence containers

Sequence containers implement data structures which can be accessed sequentially.

array (since C++11)	static contiguous array (class template)
vector	dynamic contiguous array (class template)
deque	double-ended queue (class template)
forward_list (since C++11)	singly-linked list (class template)
list	doubly-linked list (class template)

Associative containers

Associative containers implement sorted data structures that can be quickly searched ($O(\log n)$ complexity).

set	collection of unique keys, sorted by keys (class template)
map	collection of key-value pairs, sorted by keys, keys are unique (class template)
multiset	collection of keys, sorted by keys (class template)
multimap	collection of key-value pairs, sorted by keys (class template)

Unordered associative containers

Unordered associative containers implement unsorted (hashed) data structures that can be quickly searched ($O(1)$ amortized, $O(n)$ worst-case complexity).

unordered_set (since C++11)	collection of unique keys, hashed by keys (class template)
unordered_map (since C++11)	collection of key-value pairs, hashed by keys, keys are unique (class template)
unordered_multiset (since C++11)	collection of keys, hashed by keys (class template)
unordered_multimap (since C++11)	collection of key-value pairs, hashed by keys (class template)

Container adaptors

Container adaptors provide a different interface for sequential containers.

stack	adapts a container to provide stack (LIFO data structure) (class template)
queue	adapts a container to provide queue (FIFO data structure) (class template)
priority_queue	adapts a container to provide priority queue (class template)

Iterator invalidation

This section is incomplete

Thread safety

1. All container functions can be called concurrently by different threads on different containers. More generally, the C++ standard library functions do not read objects accessible by other threads unless those objects are directly or indirectly accessible via the function arguments, including the this pointer.
2. All `const` member functions can be called concurrently by different threads on the same container. In addition, the member functions `begin()`, `end()`, `rbegin()`, `rend()`, `front()`, `back()`, `data()`, `find()`, `lower_bound()`, `upper_bound()`, `equal_range()`, `at()`, and, except in associative containers, `operator[]`, behave as `const` for the purposes of thread safety (that is, they can also be called concurrently by different threads on the same container). More generally, the C++ standard library functions do not modify objects unless those objects are accessible, directly or indirectly, via the function's non-const arguments, including the this pointer.
3. Different elements in the same container can be modified concurrently by different threads, except for the elements of `std::vector<bool>` (for example, a vector of `std::future` objects can be receiving values from multiple threads). (since C++11)
4. Iterator operations (e.g. incrementing an iterator) read, but do not modify the underlying container, and may be executed concurrently with operations on other iterators on the same container, with the const member functions, or reads from the elements. Container operations that invalidate any iterators modify the container and cannot be executed concurrently with any operations on existing iterators even if those iterators are not invalidated.
5. Elements of the same container can be modified concurrently with those member functions that are not specified to access these elements. More generally, the C++ standard library functions do not read objects indirectly accessible through their arguments (including other elements of a container) except when required by its specification.
6. In any case, container operations (as well as algorithms, or any other C++ standard library functions) may be parallelized internally as long as this does not change the user-visible results (e.g. `std::transform` may be parallelized, but not `std::for_each` which is specified to visit each element of a sequence in order)

Member function table

- functions present in C++03
- functions present since C++11

		Sequence containers					Associative containers				Unordered associative containers			
Header		<array>	<vector>	<deque>	<forward_list>	<list>	<set>		<map>		<unordered_set>		<unordered_map>	
Container		array	vector	deque	forward_list	list	set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map	unordered_m
	(constructor)	(implicit)	vector	deque	forward_list	list	set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map	unordered_m
	(destructor)	(implicit)	~vector	~deque	~forward_list	~list	~set	~multiset	~map	~multimap	~unordered_set	~unordered_multiset	~unordered_map	~unordered_
	operator=	(implicit)	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operato
	assign		assign	assign	assign	assign								
Iterators	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begi
	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegi
	end	end	end	end	end	end	end	end	end	end	end	end	end	end
	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend	cend
	rbegin	rbegin	rbegin	rbegin		rbegin	rbegin	rbegin	rbegin	rbegin				
	crbegin	crbegin	crbegin	crbegin		crbegin	crbegin	crbegin	crbegin	crbegin				
	rend	rend	rend	rend		rend	rend	rend	rend	rend				
	crend	crend	crend	crend		crend	crend	crend	crend	crend				
Element access	at	at	at	at					at				at	
	operator[]	operator[]	operator[]	operator[]					operator[]				operator[]	
	front	front	front	front	front	front								
	back	back	back	back		back								
Capacity	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty	empty
	size	size	size	size		size	size	size	size	size	size	size	size	size
	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_size	max_si
	resize		resize	resize	resize	resize								
	capacity		capacity											
	reserve		reserve								reserve	reserve	reserve	reser
	shrink_to_fit		shrink_to_fit	shrink_to_fit										
Modifiers	clear		clear	clear	clear	clear	clear	clear	clear	clear	clear	clear	clear	clea
	insert		insert	insert	insert_after	insert	insert	insert	insert	insert	insert	insert	insert	inser
	emplace		emplace	emplace	emplace_after	emplace	emplace	emplace	emplace	emplace	emplace	emplace	emplace	empla
	emplace_hint						emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_
	erase		erase	erase	erase_after	erase	erase	erase	erase	erase	erase	erase	erase	erase
	push_front			push_front	push_front	push_front								
	emplace_front			emplace_front	emplace_front	emplace_front								
	pop_front			pop_front	pop_front	pop_front								
	push_back		push_back	push_back		push_back								
	emplace_back		emplace_back	emplace_back		emplace_back								
	pop_back		pop_back	pop_back		pop_back								
	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap
List operations	merge				merge	merge								
	splice				splice_after	splice								
	remove				remove	remove								
	remove_if				remove_if	remove_if								
	reverse				reverse	reverse								
	unique				unique	unique								
	sort				sort	sort								
Lookup	count						count	count	count	count	count	count	count	count
	find						find	find	find	find	find	find	find	find
	lower_bound						lower_bound	lower_bound	lower_bound	lower_bound				
	upper_bound						upper_bound	upper_bound	upper_bound	upper_bound				
	equal_range						equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_range	equal_ra
Observers	key_comp						key_comp	key_comp	key_comp	key_comp				
	value_comp						value_comp	value_comp	value_comp	value_comp				
	hash_function										hash_function	hash_function	hash_function	hash_func
	key_eq										key_eq	key_eq	key_eq	key_e
Allocator	get_allocator		get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allo
Container		array	vector	deque	forward_list	list	set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map	unordered_m
		Sequence containers					Associative containers				Unordered associative containers			

(A PDF version of this table is available at File:container-library-overview-2012-12-27.pdf.)