Lecture #16 | Standard Template Library: iterator

SE271 Object-oriented Programming (2017)

Prof. Min-gyu Cho

Previously in Object-Oriented Programming

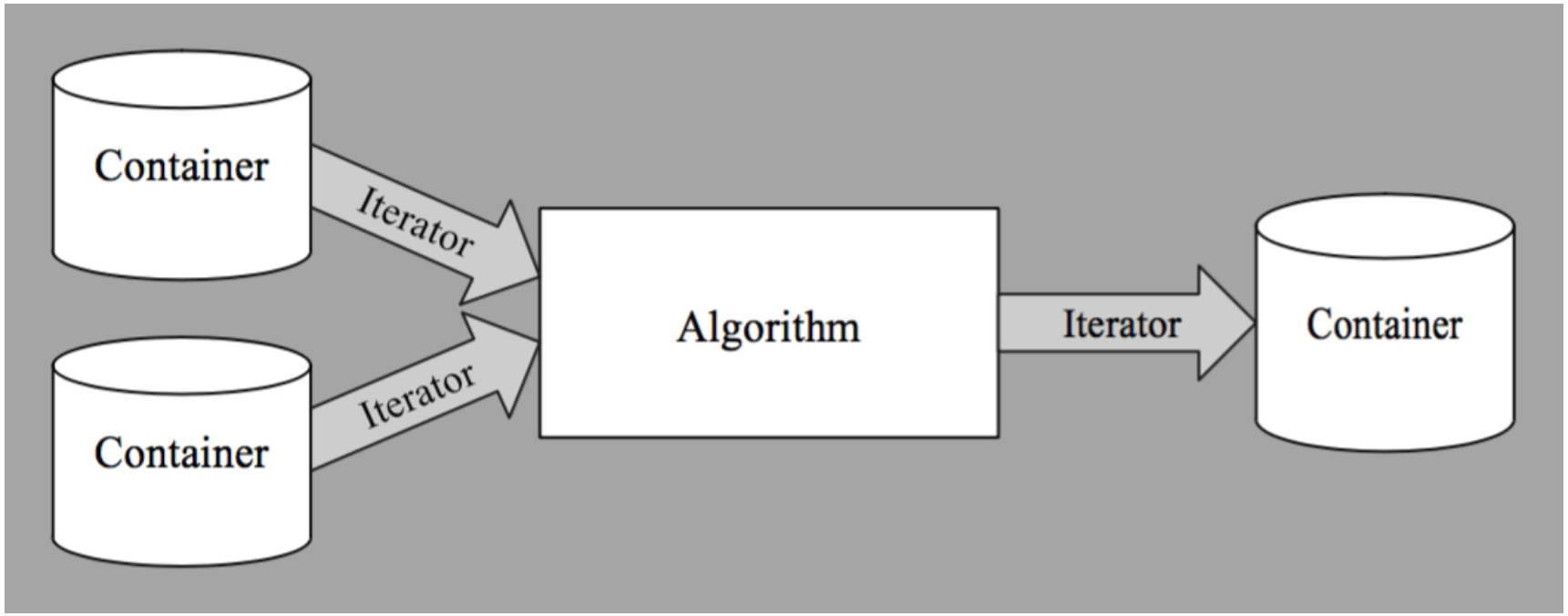
- C++ Standard Library
  - Standard Template Library (STL)
    - Container: `vector`, `list`

Today’s Topic

- Iterator
- Container: `map`, `unordered_map`

Recap: STL components and interactions

- *Containers* manage collection of objects (e.g., `vector`, `list`, `map`)
- *Algorithms* process the elements in containers (e.g., `sort`, `search`)
- *Iterators* step through the elements in containers



Before discussing iterator… Array and for loop

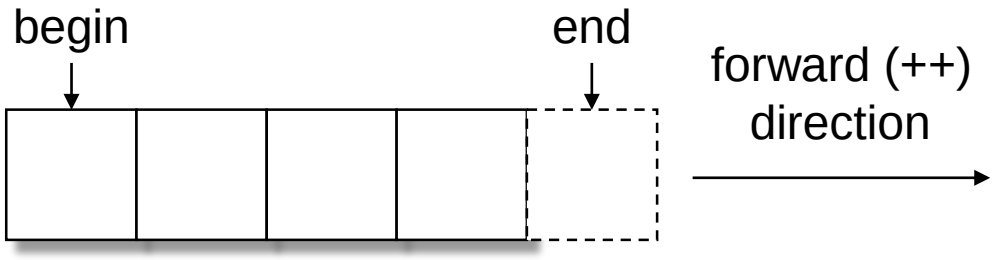
```
int array[] {42, 23, 6};
int size = sizeof(array) / sizeof(int);

int* start = array;
int* end = array + size;

for (int i = 0; i < size; ++i)
    cout << array[i] << " ";
cout << endl;

for (int* ptr = start; ptr != end; ++ptr)
    cout << *ptr << " ";
cout << endl;
```

Iterator

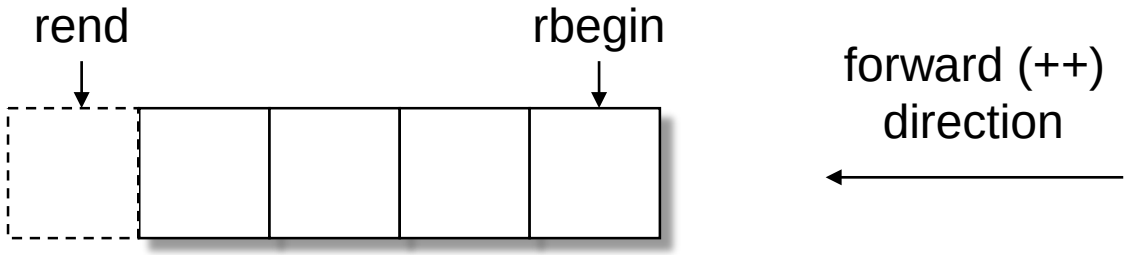
- A pointer-like object with the following operators
  - `T(a)`: copy iterator (copy constructor)
  - `++, --`: increment/decrement to indicate the next/previous object in a given container
  - `*, ->`: dereference (i.e., the object indicated by the iterator)
  - `==, !=`: test for equality or inequality
  - c.f., no assignment operator
- Iterator provides common/similar interface to access various containers

Iterator model

- Iterators give access from the beginning (inclusive) to the end (exclusive) of elements; `[begin, end)`
  - *`container`*`::iterator`



  - *`container`*`::revere_iterator`



- Note: different container types

Example: iterator

```
vector<int> vec;
list<int> lst;
for (int i = 0; i < 3; i++) {
    vec.push_back(i);
    lst.push_back(i);
}

for (vector<int>::iterator it = vec.begin(); it != vec.end(); ++it)
    cout << *it << " ";
cout << endl;
for (list<int>::iterator it = lst.begin(); it != lst.end(); ++it)
    cout << *it << " ";
cout << endl;
```

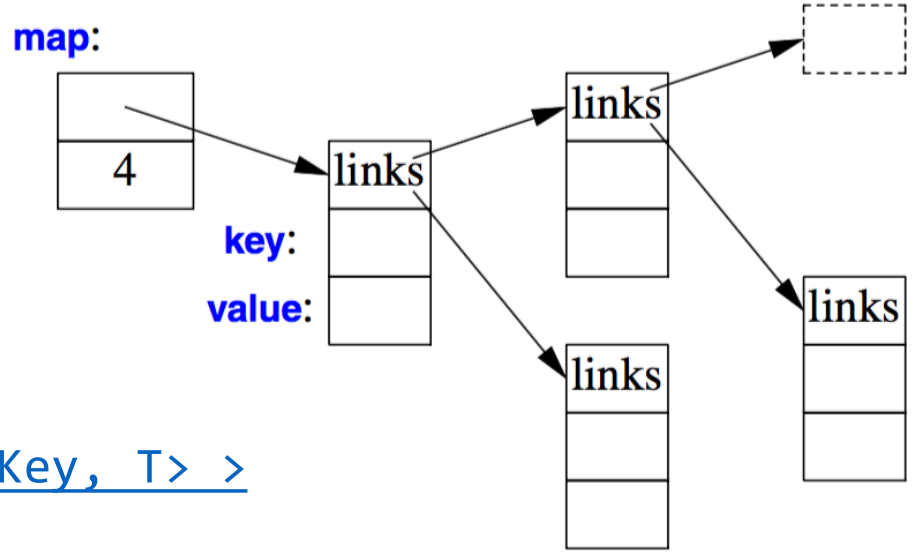
Example: reverse_iterator

```
// vector
for (vector<int>::reverse_iterator it = vec.rbegin(); it != vec.rend(); ++it)
    cout << *it << " ";
cout << endl;

// list
for (list<int>::reverse_iterator it = lst.rbegin(); it != lst.rend(); ++it)
    cout << *it << " ";
cout << endl;
```

map

- A sorted associative container, i.e., it stores key-value pairs with unique key
- Keys are sorted with comparison function
  – To use user-defined type as a key, you need to provide compare function
- Relatively cheap to access, add, remove an element

```
template<typename Key,
         typename T,
         typename Compare =
             std::less<Key>,
         typename Allocator =
             std::allocator<std::pair<const Key, T> >
> class map;
```

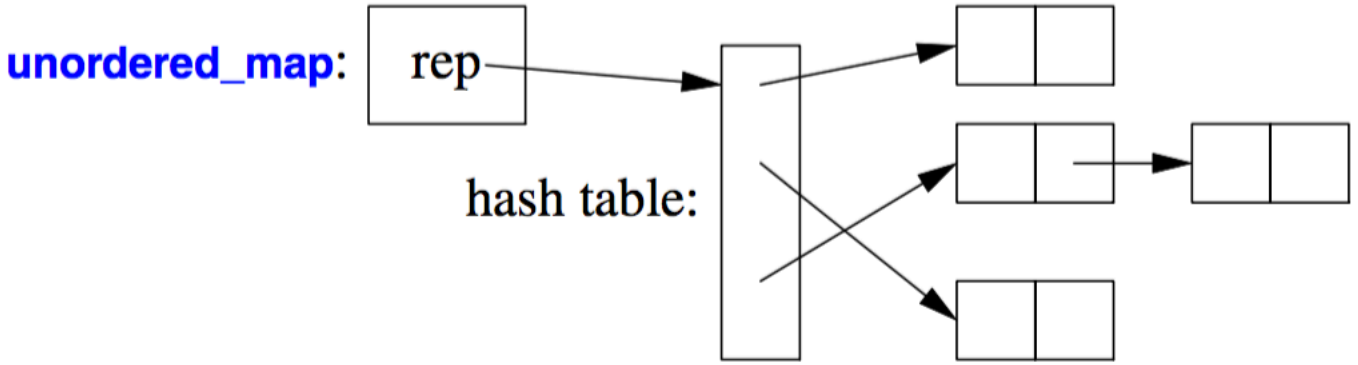


Example: map

```
map<string, int> m;
m.insert(pair<string, int>("Answer", 42));
m.insert(pair<string, int>("Birthday", 23));
m.insert(pair<string, int>("Perfect", 6));

for (map<string, int>::iterator it = m.begin(); it != m.end(); ++it)
    cout << it->first << ": " << it->second << endl;
```

unordered_map

- An associative container that contains key-value pairs with unique keys
  – To use user-defined type as a key, you need to provide hash and equal function
- Constant time for search, insertion, removal of an element




```
template<
    class Key,
    class T,
    class Hash = std::hash<Key>,
    class KeyEqual = std::equal_to<Key>,
    class Allocator = std::allocator< std::pair<const Key, T> >
> class unordered_map;
```

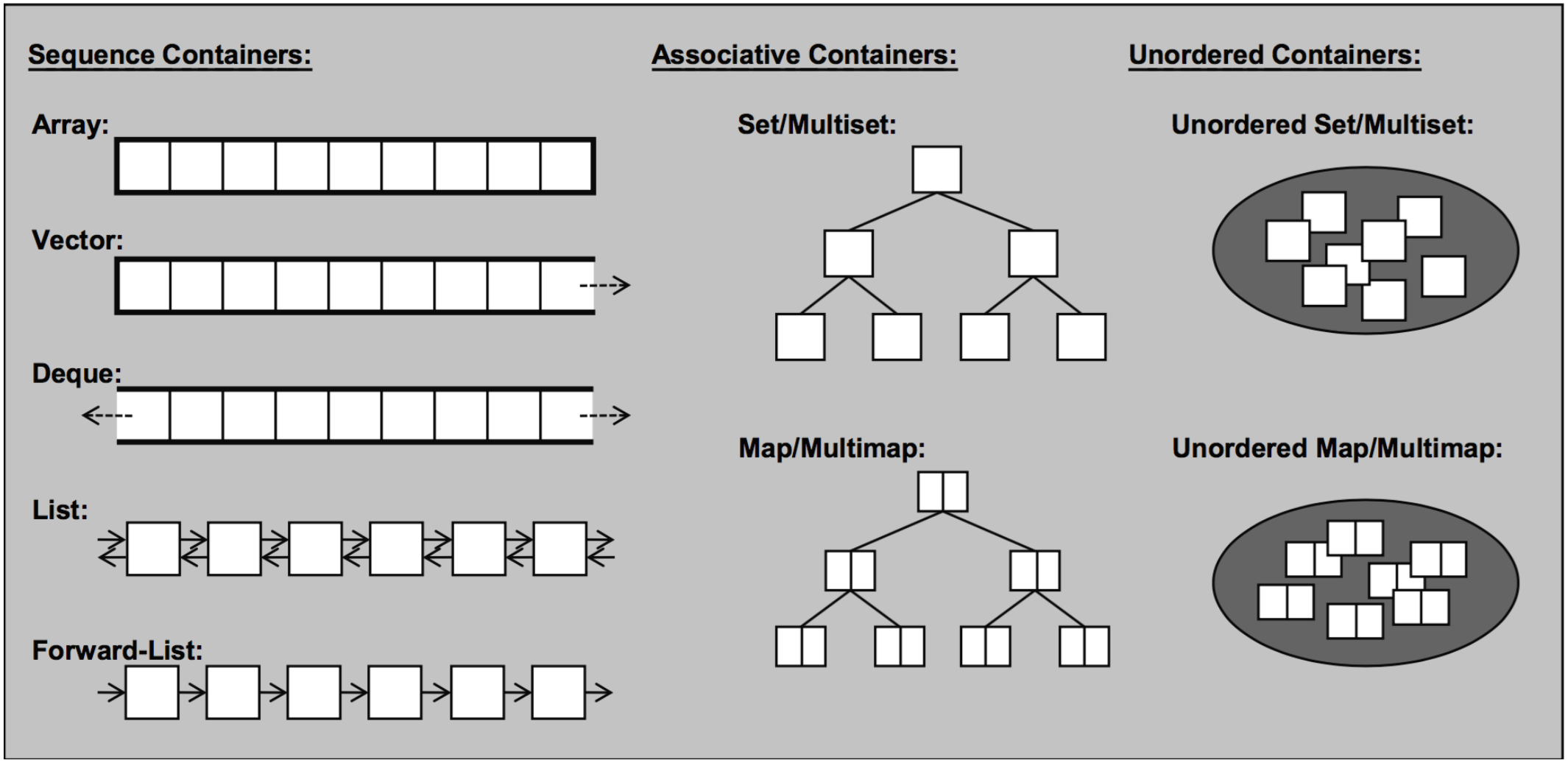
Example: unordered_map

```
unordered_map<string, int> um;
um.insert(pair<string, int>("Answer", 42));
um.insert(pair<string, int>("Birthday", 23));
um.insert(pair<string, int>("Perfect", 6));

for (unordered_map<string, int>::iterator it = um.begin(); it != um.end();
++it)
    cout << it->first << ": " << it->second << endl;
```

Standard Container Summary

Container	Internal data structure
`vector<T>`	A variable-size vector
`list<T>`	A doubly-linked list
`forward_list<T>`	A singly-linked list
`deque<T>`	A double-ended queue
`set<T>`	A set (a map with just a key and no value)
`multiset<T>`	A set in which a value can occur many times
`map<K,V>`	An associative array (§9.4)
`multimap<K,V>`	A map in which a key can occur many times
`unordered_map<K,V>`	A map using a hashed lookup (§9.5)
`unordered_multimap<K,V>`	A multimap using a hashed lookup
`unordered_set<T>`	A set using a hashed lookup
`unordered_multiset<T>`	A multiset using a hashed lookup

Standard Container Summary (Visualized)



* From “The C++ Standard Library: A tutorial and Reference (2nd edition)”

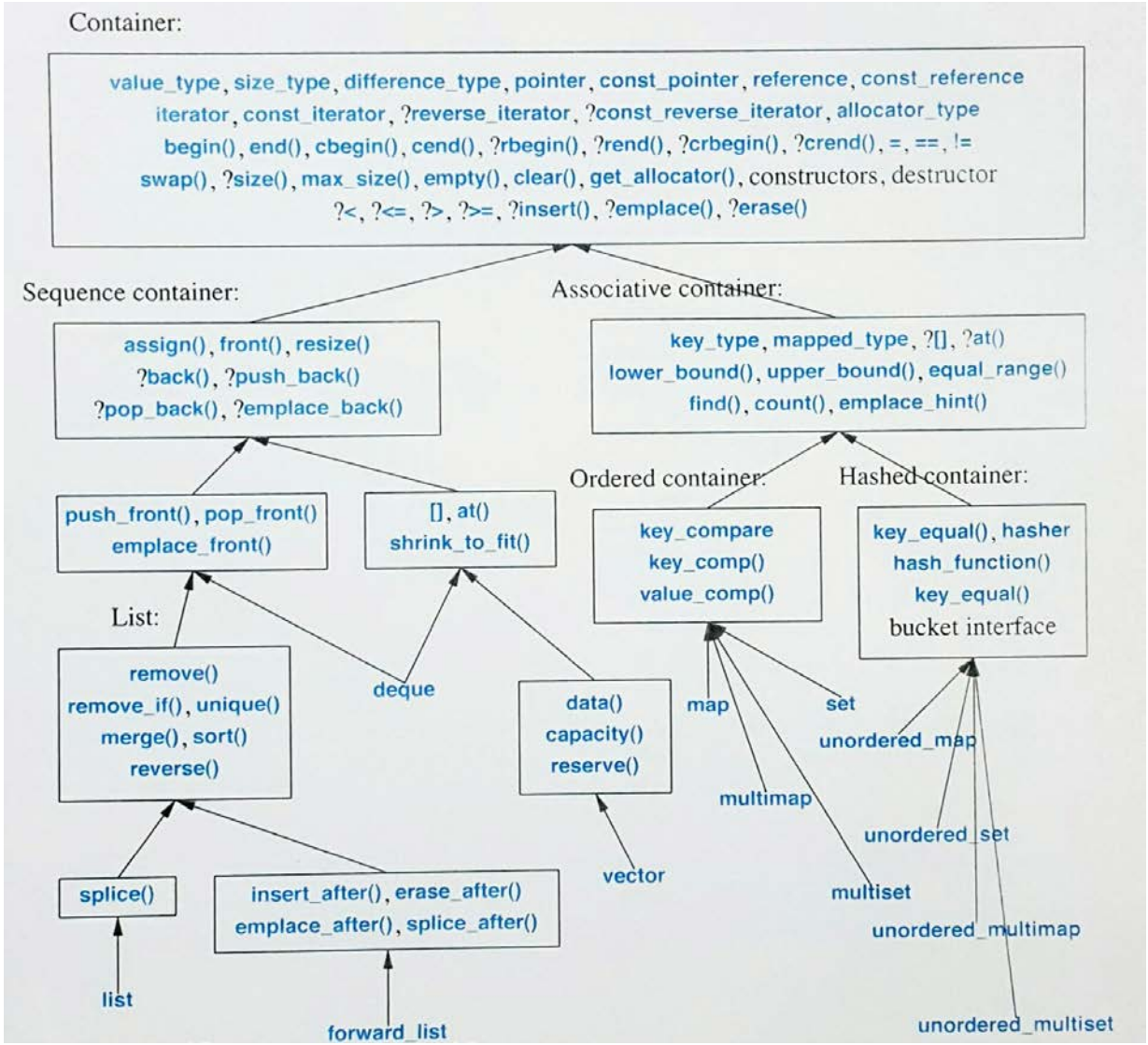
Standard Container Summary (complexity)

Container	Insertion	Access	Erase	Find	Persistent Iterators
vector / string	Back: O(1) or O(n) Other: O(n)	O(1)	Back: O(1) Other: O(n)	Sorted: O(log n) Other: O(n)	No
deque	Back/Front: O(1) Other: O(n)	O(1)	Back/Front: O(1) Other: O(n)	Sorted: O(log n) Other: O(n)	Pointers only
list / forward_list	Back/Front: O(1) With iterator: O(1) Index: O(n)	Back/Front: O(1) With iterator: O(1) Index: O(n)	Back/Front: O(1) With iterator: O(1) Index: O(n)	O(n)	Yes
set / map	O(log n)	-	O(log n)	O(log n)	Yes
unordered_set / unordered_map	O(1) or O(n)	O(1) or O(n)	O(1) or O(n)	O(1) or O(n)	Pointers only
priority_queue	O(log n)	O(1)	O(log n)	-	-

Common operators for STL containers

Function	Description
`T()`	create empty container (default constructor)
`T(const T&)`	copy container (copy constructor)
`T(T&&)`	move container (move constructor)
`~T()`	destroy container (including its elements)
`empty()`	test if container empty
`size()`	get number of elements in container
`push_back()`	insert an element at end of container (sequential)
`insert()`	insert an element (associative/unordered)
`clear()`	remove all elements from container
`operator=()`	assign all elements of one container to other
`operator[]()`	access element in container

Operations of STL containers



* From “The C++ programming language”

Operations on containers

		Sequence containers					Associative containers				Unordered associative containers				Container adaptors		
Headers		<array>	<vector>	<deque>	<forward_list>	<list>	<set>		<map>		<unordered_set>		<unordered_map>		<stack>	<queue>	
		array	**vector**	**deque**	**forward_list**	**list**	**set**	**multiset**	**map**	**multimap**	**unordered_set**	**unordered_multiset**	**unordered_map**	**unordered_multimap**	**stack**	**queue**	**priority_queue**
	(constructor)	(implicit)	vector	deque	forward_list	list	set	multiset	map	multimap	unordered_set	unordered_multiset	unordered_map	unordered_multimap	stack	queue	priority_queue
	(destructor)	(implicit)	~vector	~deque	~forward_list	~list	~set	~multiset	~map	~multimap	~unordered_set	~unordered_multiset	~unordered_map	~unordered_multimap	~stack	~queue	~priority_queue
	operator=	(implicit)	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=
	assign		assign	assign	assign	assign											
Iterators	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin			
	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin			
	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										front	top
	back	back	back	back		back									top	back	
Capacity	empty	empty	empty	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	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_size			
	resize		resize	resize	resize	resize											
	capacity		capacity														
	reserve		reserve								reserve	reserve	reserve	reserve			
	shrink_to_fit		shrink_to_fit	shrink_to_fit													
Modifiers	clear		clear	clear	clear	clear	clear	clear	clear	clear	clear	clear	clear	clear			
	insert		insert	insert	insert_after	insert	insert	insert	insert	insert	insert	insert	insert	insert			
	emplace		emplace	emplace	emplace_after	emplace	emplace	emplace	emplace	emplace	emplace	emplace	emplace	emplace			
	emplace_hint						emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint	emplace_hint			
	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										pop	
	push_back		push_back	push_back		push_back									push	push	push
	emplace_back		emplace_back	emplace_back		emplace_back									emplace	emplace	emplace
	pop_back		pop_back	pop_back		pop_back									pop		pop
	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap	swap
List operations	merge					merge											
	splice					splice											
	remove					remove											
	remove_if					remove_if											
	reverse					reverse											
	unique					unique											
	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_range			
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_function			
	key_eq										key_eq	key_eq	key_eq	key_eq			
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_allocator			
		array	**vector**	**deque**	**forward_list**	**list**	**set**	**multiset**	**map**	**multimap**	**unordered_set**	**unordered_multiset**	**unordered_map**	**unordered_multimap**	**stack**	**queue**	**priority_queue**
		Sequence containers					Associative containers				Unordered associative containers				Container adaptors		

C++ Containers Library cross-reference table from http://en.cppreference.com/w/cpp/container PDF version with red & orange lines by Robin Whittle December 2012.

- functions present since C++11

- functions present in C++03

* http://en.cppreference.com/w/cpp/container

Operations on containers (part 1)

		Sequence containers					Associative containers			
Headers		<array>	<vector>	<deque>	<forward_list>	<list>	<set>		<map>	
		array	vector	deque	forward_list	list	set	multiset	map	multimap
	(constructor)	(implicit)	vector	deque	forward_list	list	set	multiset	map	multimap
	(destructor)	(implicit)	~vector	~deque	~forward_list	~list	~set	~multiset	~map	~multimap
	operator=	(implicit)	operator=	operator=	operator=	operator=	operator=	operator=	operator=	operator=
	assign		assign	assign	assign	assign				
Iterators	begin	begin	begin	begin	begin	begin	begin	begin	begin	begin
	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin	cbegin
	end	end	end	end	end	end	end	end	end	end
	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	
	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
	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
	resize		resize	resize	resize	resize				
	capacity		capacity							
	reserve		reserve							
	shrink_to_fit		shrink_to_fit	shrink_to_fit						

* http://en.cppreference.com/w/cpp/container

Operations on containers (part 2)

Modifiers	clear		clear	clear	clear	clear	clear	clear	clear	clear
	insert		insert	insert	insert_after	insert	insert	insert	insert	insert
	emplace		emplace	emplace	emplace_after	emplace	emplace	emplace	emplace	emplace
	emplace_hint						emplace_hint	emplace_hint	emplace_hint	emplace_hint
	erase		erase	erase	erase_after	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
List operations	merge					merge				
	splice					splice				
	remove					remove				
	remove_if					remove_if				
	reverse					reverse				
	unique					unique				
	sort					sort				
Lookup	count						count	count	count	count
	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
Observers	key_comp						key_comp	key_comp	key_comp	key_comp
	value_comp						value_comp	value_comp	value_comp	value_comp
	hash_function									
	key_eq									
Allocator	get_allocator		get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator	get_allocator
		array	**vector**	**deque**	**forward_list**	**list**	**set**	**multiset**	**map**	**multimap**
		Sequence containers					Associative containers			

* http://en.cppreference.com/w/cpp/container

Reference: categories of iterator

Category	Capability	Provider
Input	Read (once only) forward	`istream (istream_iterator)`
Output	Write (once only) forward	`ostream (ostream_iterator), inserter_iterator`
Forward	Read and write forward	`forward_list, unordered_set, unordered_map`
Bidirectional	Read and write forward and backward	`list, set, multiset, map, multimap`
Random access	Read and write with random access	`array, vector, string`

Reference: input iterator

Operator	Behavior
`T(a)`	copies iterator (copy constructor)
`*a` `a->m`	dereference as rvalue (i.e., read only); can only be dereferenced once
`++a`	steps forward (returns new position)
`a++`	steps forward (returns old position)
`a == b`	test for equality
`a != b`	test for inequality

- Not assignable

Reference: output iterator

Operator	Behavior
`T(a)`	copies iterator (copy constructor)
`*a`	dereference as rvalue (i.e., read only);
`a->m`	can only be dereferenced once
`++a`	steps forward (returns new position)
`a++`	steps forward (returns old position)

- Not assignable
- No comparison operators (i.e., no `operator==`, `operator!=`)

Reference: forward iterator

Operator	Behavior
`T()`	default constructor
`T(a)`	copy constructor
`a = b`	assignment
`*a` `a->m`	dereference as lvalue (i.e., write only); can only be dereferenced once
`++a`	steps forward (returns new position)
`a++`	steps forward (returns old position)
`a == b`	test for equality
`a != b`	test for inequality

- Must ensure that valid to dereference iterator before doing so

Reference: forward iterator

Operator	Behaviour
`T()`	default constructor
`T(a)`	copy constructor
`a = b`	assignment
`*a` `a->m`	dereference as lvalue (i.e., write only); can only be dereferenced once
`++a`	steps forward (returns new position)
`a++`	steps forward (returns old position)
`a == b`	test for equality
`a != b`	test for inequality

- Must ensure that valid to dereference iterator before doing so

Reference: bidirectional iterator

Operator	Behavior
`--a`	steps backward (returns new position)
`a--`	steps backward (returns old position)

- Provide all the operations of forward iterators & the 2 backward operators
  - Iterators can move both forward and backward

Reference: random-access iterator

Operator	Behavior
a[n]	deference element at index n (where n can be negative)
a += n	steps n elements forward (where n can be negative)
a -= n	steps n elements backward (where n can be negative)
a + n	iterator for n-th next element
n + a	iterator for n-th next element
a - n	iterator for n-th previous element
a - b	distance from a to b
a < b	test if a before b
a > b	test if a after b
a <= b	test if a not after b
a >= b	test if a not before b

- Provide all the operations of bidirectional & random-access operators

Reading list

- Learn C++
  - C++ standard library: Ch. 9.8
- STL containers
  - List of member functions: http://en.cppreference.com/w/cpp/container
- (Advanced)
  - LLVM C++ standard library implementation: http://libcxx.llvm.org
    - Follow ‘building libc++’ link for download

ANY QUESTIONS?