C++11 STL additions

Algorithms

bool **all_of**(Iter first, Iter last, Pred pred)	true if all the values in [first, last) satisfy the predicate (or the range is empty), false otherwise
bool **any_of**(Iter first, Iter last, Pred pred)	true if at least one of the values in [first, last) satisifes the predicate, false otherwise (or if the range is empty)
bool **none_of**(Iter first, Iter last, Pred pred)	true if no values in [first, last) satisfy the predicate (or if the range is empty), false otherwise
Iter **find_if_not**(Iter first, Iter last, Pred pred)	returns the first iterator i in the range where pred(*i) == false or last if no such iterator found
OutIter **copy_if**(InIter first, InIter last, OutIter result, Pred pred)	copy all elements in [first, last) that satisfy a predicate into a range starting from result (the opposite of remove_copy_if)
OutIter **copy_n**(InIter first, Size n, OutIter result)	copies n elements starting from first into a range starting from result
uninitialized_copy_n(InIter first, Size n, OutIter result)	invokes uninitialized_copy for n elements
OutIter **move**(InIter first, InIter last, OutIter result)	moves elements from [first, last) into a range starting from result
OutIter **move_backward**(InIter first, InIter last, OutIter result)	moves elements in the range [first, last) into the range [result - (last - first), result) starting from last - 1 and proceeding to first
is_partitioned(InIter first, InIter last, Pred pred)	true if [first, last) is empty or if [first, last) is partitioned by pred, i.e. if all elements that satisfy pred appear before those that don't
pair<OutIter1, OutIter2> **partition_copy**(InIter first, InIter last, OutIter1 out_true, OutIter2 out_false, Pred pred)	copies elements that satisfy pred from [first, last) into the range starting with out_true, and other elements into the range starting with out_false
Iter **partition_point**(Iter first, Iter last, Pred pred)	returns an iterator to the 1st element in [first, last) that doesn't satisfy pred
RAIter **partial_sort_copy**(InIter first, InIter last, RAIter result_first, RAIter result_last, Compare comp)	copies sorted elements from [first, last) into the result range (in terms of comp If supplied); the number of elements copied is determined by the size of the smaller of input and result ranges
bool **is_sorted**(Iter first, Iter last) bool **is_sorted**(Iter first, Iter last, Compare comp)	true if [first, last) is sorted (in terms of comp if supplied), false otherwise
Iter **is_sorted_until**(Iter first, Iter last) Iter **is_sorted_until**(Iter first, Iter last, Compare comp)	returns the last iterator i in [first, last] for which the range [first, i) is sorted (in terms of comp if supplied)
bool **is_heap**(Iter first, Iter last) bool **is_heap**(Iter first, Iter last, Compare comp)	true if [first, last) is a heap (in terms of comp if supplied), i.e. the first element is the largest
Iter **is_heap_until**(Iter first, Iter last) Iter **is_heap_until**(Iter first, Iter last, Compare comp)	returns the last iterator i in [first, last] for which the range [first, i) is a heap (in terms of comp if supplied)
T **min**(initializer_list<T> t) T **min**(initializer_list<T> t, Compare comp)	returns the smallest value (in terms of comp if supplied) in the initializer_list
T **max**(initializer_list<T> t) T **max**(initializer_list<T> t, Compare comp)	returns the largest value in the initializer_list (in terms of comp if supplied)
pair<const T&, const T&> **minmax**(const T& a, const T& b) pair<const T&, const T&> **minmax**(const T& a, const T& b, Compare comp)	returns (b, a) pair if b < a (in terms of comp if supplied), and (a, b) pair otherwise
pair<const T&, const T&> **minmax**(initializer_list<T> t) pair<const T&, const T&> **minmax**(initializer_list<T> t, Compare comp)	returns the smallest and the largest element in initializer_list (in terms of comp if supplied)
pair<Iter, Iter> **minmax_element**(Iter first, Iter last) pair<Iter, Iter> **minmax_element**(Iter first, Iter last, Compare comp)	returns the first iterator in [first, last) pointing to the smallest element, and the last iterator pointing to the largest element (in terms of comp if supplied)
void **iota**(Iter first, Iter last, T value)	creates a range of sequentially increasing values; assigns *i = value to each element in [first, last) and increments value as if by ++value

Containers

unordered_set<T> contains at most one of each value and provides fast retrieval of values; supports forward iterators

General functions	Modifiers	Bucket functions
operator=	clear	begin(int)
get_allocator	insert	end(int)
	emplace	bucket_count
Iterators	emplace_hint	max_bucket_count
begin/cbegin	erase	bucket_size
end/cend	swap	bucket
Capacity	**Lookup**	**Hash policy**
erase	count	load_factor
size	find	max_load_factor
max_size	equal_range	rehash
		reserve
Observers		
hash_function		
key_eq		

unordered_multiset<T> supports equivalent values (possibly with multiple copies of the same value) and provides fast retrieval of the values; supports forward iterators

General functions	Modifiers	Bucket functions
operator=	clear	begin(int)
get_allocator	insert	end(int)
	emplace	bucket_count
Iterators	emplace_hint	max_bucket_count
begin/cbegin	erase	bucket_size
end/cend	swap	bucket
Capacity	**Lookup**	**Hash policy**
erase	count	load_factor
size	find	max_load_factor
max_size	equal_range	rehash
		reserve
Observers		
hash_function		
key_eq		

unordered_map<Key, T> hash table; contains at most one of each key value; supports forward iterators

General functions	Modifiers	Bucket functions
operator=	clear	begin(int)
get_allocator	insert	end(int)
	emplace	bucket_count
Iterators	emplace_hint	max_bucket_count
begin/cbegin	erase	bucket_size
end/cend	swap	bucket
Capacity	**Lookup**	**Hash policy**
erase	count	load_factor
size	find	max_load_factor
max_size	equal_range	rehash
		reserve
Observers		
hash_function		
key_eq		

unordered_multimap<Key, T> hash table; supports equivalent keys (can contain multiple copies of each key value); supports forward iterators

General functions	Modifiers	Bucket functions
operator=	clear	begin(int)
get_allocator	insert	end(int)
	emplace	bucket_count
Iterators	emplace_hint	max_bucket_count
begin/cbegin	erase	bucket_size
end/cend	swap	bucket
Capacity	**Lookup**	**Hash policy**
erase	count	load_factor
size	find	max_load_factor
max_size	equal_range	rehash
		reserve
Observers		
hash_function		
key_eq		

forward_list<T> singly linked list; constant time insert and erase operations; automatic storage management; no fast random access

General functions	Capacity	Modifiers
operator=	empty	clear
assign	max_size	insert_after
get_allocator		emplace_after
	Operations	erase_after
Element access	merge	push_front
front	splice_after	emplace_front
	remove	pop_front
Iterators	remove_if	resize
before_begin/ cbefore_begin	reverse	swap
begin/cbegin	unique	
end/cend	sort	

array<T, N> stores fixed size sequences of objects (N elements of type T); elements are stored contiguously

Element access	Capacity
at	
operator[]	empty
front	size
back	max_size
data	
Iterators	**Modifiers**
begin/cbegin	fill
end/cend	swap
rbegin/crbegin	
rend/crend	

Want to be a C++11 expert?
Check out cpprocks.com