Containers

The STL contains sequence containers and associative containers. The standard sequence containers include vector, deque, and list. The standard associative containers are set, multiset, map, and multimap. There are also *container adaptors* queue, priority_queue, and stack, that are containers with specific interface, using other containers as implementation.

Container	Description
Simple Containers	
pair	The pair container is a simple associative container consisting of a 2-tuple of data elements or objects, called 'first' and 'second', in that fixed order. The STL 'pair' can be assigned, copied and compared. The array of objects allocated in a map or hash_map (described below) are of type 'pair' by default, where all the 'first' elements act as the unique keys, each associated with their 'second' value objects.
Sequences (Arrays/Linked Lists): ordered collections	
vector	a dynamic array, like C array (i.e., capable of random access) with the ability to resize itself automatically when inserting or erasing an object. Inserting and removing an element to/from back of the vector at the end takes amortized constant time. Inserting and erasing at

	the beginning or in the middle is linear i time. A specialization for type bool exists, which optimizes for space by storing bool values as bits.
list	a doubly linked list; elements are not stored in contiguous memory. Opposite performance from a vector. Slow lookup and access (linear time), but once a position has been found, quick insertion and deletion (constant time).
deque (***double-ended*queue**)	a vector with insertion/erase at the beginning or end in amortized constant time, however lacking some guarantees on iterator validity after altering the deque.
Container adaptors	
queue	Provides FIFO queue interface in terms of push/pop/front/back operations. Any sequence supporting operations front(), back(), push_back(), and pop_front() can be used to instantiate queue (e.g. list and deque).
priority queue	Provides priority queue interface in terms of push/pop/top operations

	(the element with the highest priority is on top). Any random-access sequence supporting operations front(), push_back(), and pop_back() can be used to instantiate priority_queue (e.g. vector and deque). Elements should additionally support comparison (to determine which element has a higher priority and should be popped first).
stack	Provides LIFO stack interface in terms of push/pop/top operations (the last-inserted element is on top). Any sequence supporting operations back(), push_back(), and pop_back() can be used to instantiate stack (e.g. vector, list, and deque).
Associative containers: unordered collections	
set	a mathematical set; inserting/erasing elements in a set does not invalidate iterators pointing in the set. Provides se operations union, intersection,difference, symmetric difference and test of inclusion. Type of

	data must implement comparison operator < or custom comparator function must be specified; such comparison operator or comparator function must guarantee strict weak ordering, otherwise behavior is undefined. Typically implemented using a self-balancing binary search tree.
multiset	same as a set, but allows duplicate elements (mathematical Multiset).
map	an associative array; allows mapping from one data item (a key) to another (a value). Type of key must implement comparison operator < or custom comparator function must be specified; such comparison operator or comparator function must guarantee strict weak ordering, otherwise behavio is undefined. Typically implemented using a self-balancing binary search tree.
multimap	same as a map, but allows duplicate keys.
unordered_set unordered_multiset unordered_map	similar to a set, multiset, map, or multimap, respectively, but implemented using a hash table; keys are not ordered, but a hash function

unordered_multimap	must exist for the key type. These containers are part of C++11.
Other types of containers	
bitset	stores series of bits similar to a fixed-sized vector of bools. Implements bitwise operations and lacks iterators. Not a Sequence.
valarray	another C-like array like vector, but is designed for high speed numerics at the expense of some programming ease and general purpose use. It has many features that make it ideally suited for use with vector processors in traditional vector supercomputers and SIMD units in consumer-level scalar processors, and also ease vector mathematics programming even in scalar computers.

Iterators

The STL implements five different types of iterators. These are *input iterators* (that can only be used to read a sequence of values), *output iterators* (that can only be used to write a sequence of values), *forward iterators* (that can be read, written to, and move forward), *bidirectional iterators* (that are like forward iterators, but can also move backwards) and *random access iterators* (that can move freely any number of steps in one operation).

It is possible to have bidirectional iterators act like random access iterators, as moving forward ten steps could be done by simply moving forward a step at a time a total of ten times. However,

having distinct random access iterators offers efficiency advantages. For example, a vector would have a random access iterator, but a list only a bidirectional iterator.

Iterators are the major feature that allow the generality of the STL. For example, an algorithm to reverse a sequence can be implemented using bidirectional iterators, and then the same implementation can be used on lists, vectors and deques. User-created containers only have to provide an iterator that implements one of the five standard iterator interfaces, and all the algorithms provided in the STL can be used on the container.

This generality also comes at a price at times. For example, performing a search on an associative container such as a map or set can be much slower using iterators than by calling member functions offered by the container itself. This is because an associative container's methods can take advantage of knowledge of the internal structure, which is opaque to algorithms using iterators.

Algorithms

A large number of algorithms to perform operations such as searching and sorting are provided in the STL, each implemented to require a certain level of iterator (and therefore will work on any container that provides an interface by iterators).

Functors

The STL includes classes that overload the function call operator (`operator()`). Instances of such classes are called functors or function objects. Functors allow the behavior of the associated function to be parameterized (e.g. through arguments passed to the functor's constructor) and can be used to keep associated per-functor state information along with the function. Since both functors and function pointers can be invoked using the syntax of a function call, they are interchangeable as arguments to templates when the corresponding parameter only appears in function call contexts.

A particularly common type of functor is the predicate. For example, algorithms like `find_if` take a unary predicate that operates on the elements of a sequence. Algorithms like sort, partial_sort, nth_element and all sorted containers use a binary predicate that must provide a strict weak ordering, that is, it must behave like a membership test on a transitive, non reflexive and

asymmetric binary relation. If none is supplied, these algorithms and containers use less by default, which in turn calls the less-than-operator <.

SOURCE : WIKIPEDIA