STL Containers

The ISO Standard Template Library (STL) provides containers for storing collections of related objects. The containers are template classes that enable specification of objects that are allowed in the containers.
Containers in the STL can be divided into three categories: sequence containers, associative containers, and container adapters.

Sequence Containers

Sequence containers maintain the ordering of inserted elements that you specify.
A **vector** container behaves like an array, but can automatically grow as required. It is random access and contiguously stored, and length is highly flexible. For these reasons and more, **vector** is the preferred sequence container for most applications.
An **array** container has some of the strengths of **vector**, but the length is not as flexible.
A **deque** (double-ended queue) container allows for fast insertions and deletions at the beginning and end of the container. It shares the random-access and flexible-length advantages of **vector**, but is not contiguous.
A **list** container is a doubly linked list that enables bidirectional access, fast insertions, and fast deletions anywhere in the container, but you cannot randomly access an element in the container.
A **forward_list** container is a singly linked list—the forward-access version of **list**.

Associative Containers

In associative containers, elements are inserted in a pre-defined order—for example, as sorted ascending. Unordered associative containers are also available. The associative containers can be grouped into two subsets: maps and sets.
A **map**, sometimes referred to as a dictionary, consists of a key/value pair. The key is used to order the sequence, and the value is associated with that key. For example, a **map** might contain keys that represent every unique word in a text and corresponding values that represent the number of times that each word appears in the text. The unordered version of **map** is **unordered_map**.
A **set** is just an ascending container of unique elements—the value is also the key. The unordered version of **set** is **unordered_set**.
Both **map** and **set** only allow one instance of a key or element to be inserted into the container. If multiple instances of elements are required, use **multimap** or **multiset**. The unordered versions are **unordered_multimap** and **unordered_multiset**.

Ordered maps and sets support bi-directional iterators, and their unordered counterparts support forward iterators.

Container Adapters

A container adapter is a variation of a sequence or associative container that restricts the interface for simplicity and clarity. Container adapters do not support iterators.
A **queue** container follows FIFO (first in, first out) semantics. The first element *pushed*—that is, inserted into the queue—is the first to be *popped*—that is, removed from the queue.
A **priority_queue** container is organized such that the element that has the highest value is always first in the queue.
A **stack** container follows LIFO (last in, first out) semantics. The last element pushed on the stack is the first element popped.
Because container adapters do not support iterators, they cannot be used with the STL algorithms.

Requirements for Container Elements

In general, elements inserted into an STL container can be of just about any object type if they are copyable. Movable-only elements—for example, those such as `vector<unique_ptr<T>>` that are created by using **unique_ptr<>** will work as long as you don't call member functions that attempt to copy them.
The destructor is not permitted to throw an exception.
Ordered associative containers—described earlier in this article—must have a public comparison operator defined. (By default, the operator is **operator<**, but even types that don't work with **operator<** are supported.
Some operations on containers might also require a public default constructor and a public equivalence operator. For example, the unordered associative containers require support for equality and hashing.

Accessing Container Elements

The elements of containers are accessed by using iterators.