**Effective STL**

The STL has iterators, algorithms, and function objects. More powerful and flexible than arrays, they grow (and often shrink) dynamically, manage their own memory, keep track of how many objects they hold, bound the algorithmic complexity of the operations they support, and much, much more.

**Item 1. Choose your containers with care.**

C++ has variety of containers:

* The **standard STL sequence containers:** vector, string, deque, and list.
* The **standard STL associative containers**: set, multiset, map and multimap.
* The **nonstandard sequence containers** slist and rope.
* The **nonstandard associative containers** hash\_set, hash\_multiset, hash\_map,

and hash\_multimap.

* **vector<char> as a replacement for string.** There are certain conditions

under which such a replacement might make sense.

* **vector as a replacement for the standard associative containers. T**here are several times when vector can outperform the standard associative containers in both time and space.
* Several **standard non-STL containers,** including arrays, bitset, valarray, stack,

queue, and priority\_queue.

Case where arrays are preferable to STL containers. arrays can be used with

STL algorithms, because pointers can be used as array iterators. (Item 16)

bitset may be better than vector<bool> (Item 18)

vector, list, and deque offer the programmer different complexity trade-offs and should be used accordingly, vector is the type of sequence that should be used by default, list should be used when there are frequent insertions and deletions from the middle of the sequence, deque is the data structure of choice when most insertions and deletions take place at the beginning or at the end of the sequence.

**Contiguous-memory containers vs node-based containers**

*Contiguous-memory containers* (also known as *array-based containers]* store their

elements in one or more (dynamically allocated) chunks of memory, each chunk

holding more than one container element. If a new element is inserted or an existing

element is erased, other elements in the same memory chunk must be shifted up or

down to make room for the new element or to fill the space formerly occupied by the

erased element. This kind of movement affects both performance and exception safety. The standard contiguous-memory containers are vector, string, and deque. The nonstandard rope is also a contiguous-memory container.

*Node-based containers* store only a single element per chunk of (dynamically

allocated) memory. Insertion or erasure of a container element affects only pointers to

nodes, not the contents of the nodes themselves, so element values need not be moved

when something is inserted or erased.

* For example: link list (list, slist), standard associative containers (set, multiset, map and multimap) and nonstandard hashed containers (hash\_set, hash\_multiset, hash\_map, and hash\_multimap).

**How to choose the container based of our requirements:**

* *If we want to insert a new element at an arbitrary position in the container:* If so, then need a sequence container, associative containers won't do.
* *If we care how elements are ordered in the container?* If not. a hashed container becomes a viable choice. Otherwise, you'll want to avoid hashed containers.
* *Must the container be part of standard C++?* If so, that eliminates hashed

containers, slist, and rope.

* What *category of iterators we require?* If they must be random access iterators, then technically limited to vector, deque, and string, If bidirectional iterators are required, you must avoid slist.
* *Is it important to avoid movement of existing container elements when insertions or erasures take place?* If so, need to stay away from contiguous-memory containers.
* *Does the data in the container need to be layout-compatible with C*? If so,

we're limited to vectors.

* *Is lookup speed a critical consideration?* If so, look at hashed containers, sorted vectors, and the standard associative containers — probably in that order.
* *Do you mind if the underlying container uses reference counting? Use string.*
* Do *you need transactional semantics for insertions and erasures with reliably roll back insertions and erasures?* For a single element operation choose node-based container.If we need transactional semantics for multiple-element insertions need to choose list, because list is the only standard container that offers transactional semantics for multiple-element insertions.
* If we *need to minimize iterator, pointer, and reference invalidation?* If so,

you'll want to use node-based containers, because insertions and erasures on

such containers never invalidate iterators, pointers, or references (unless they

point to an element you are erasing). In general, insertions or erasures on

contiguous-memory containers may invalidate all iterators, pointers, and

references into the container.

* Would it be helpful to have a sequence container with random access iterators

where pointers and references to the data are not invalidated if nothing

is erased and insertions take place only at the ends of the container?

This is a very special case, but if it's your case, deque is perfect choice.

(Interestingly, deque's iterators may be invalidated when insertions are made

only at the ends of the container, deque is the only standard STL container

whose iterators may be invalidated without also invalidating its pointers and

references.)