C++

Standard containers

Containers in the standard library

Four types of containers:

- Sequences: vector, list, deque
- Associative containers: set, map, multiset, multimap
- Container adapters: stack, queue, priority_queue
- Others: strings, valarrays and bitsets

Operations summary

- Member types
- Iterators
- Element access
- Stack and queue operations
- List operations
- Constructors
- Assignments
- Associative operations
- Others (size and capacity, helper functions)

Member types

value_type	Type of element
allocator_type	Type of memory manager
size_type	Type of subscripts
difference_type	Type of difference between iterators
iterator	Like value_type*
const_iterator, reverse_iterator, const_reverse_iterator	
reference	Like value_type&
const_reference	
key_type	Type of key for associative containers (only)
mapped_type	Type of mapped_value for associative containers (only)
key_compare	Type of comparison criterion for associative containers (only)

Iterators

begin() Iterator pointing to first element
end() Iterator pointing to one past last element
rbegin() Iterator pointing to first element of reverse sequence
rend() Iterator pointing to one pas last element of reverse sequence

Element access

front() Return first element
back() Return last element
[] Access i-th element, unchecked (not for list)
at() Access i-th element, checked (not for list)

Stack and queue operations

push_back() Add to end
push_front() Add at the beginning (for list and deque only)
pop_back() Remove last element
pop_front() Remove first element (for list and deque only)

List operations

insert(p, x)	Add x before p(p is an iterator)
insert(p, n, x)	Add n copies of x before p
insert(p, first, last)	Add elements in [first; last(before p
erase(p)	Erase element at p
erase(first, last)	Erase elements in [first; last(
clear()	Erase all elements in the container

Others

size()	Number of elements in the container
empty()	Is the container empty
max_size()	Size of the largest possible container
capacity()	Space allocated (for vector only)
reserve()	Reserve memory (for vector only)
resize()	Change size (vector, list and deque)
swap()	Swap elements of two containers
get_allocator()	Get a copy of the allocator
==	Check equality of two containers' contents
!=	
<	Check lexicographic order

Constructors

container()	Construct an empty container
container(n)	n elements (not for assoc container)
container(n, x)	n copies of x (not for assoc container)
container(first, last)	Use elements from [first; last(
container(c)	Copy ctor: use elements from c
~container()	

Assignments

operator=(c)	Assign copy of elements from c
assign(n, x)	Assign n copies of x (not assoc. container)
assign(first, last)	Assign copy from [first; last(

Lookup operations (for associative containers)

operator[](k)	Access element with key k
find(k)	Find element with key k
lower_bound(k)	Find first element with key k
upper_bound(k)	Find first element with key greater than k
key_comp()	Copy of the comparison op for keys
value_comp()	Copy of the comparison op for mapped_value

Time complexity and container selection

- Standard containers have common operations that make them interchangeable
- Selection of a container for a task depends on the need of a specialized operation and on efficiency reason
- Example:
  - if insertion/deletion inside a container will be frequent, then a list should be chosen
  - if we need to lookup based on a key, then a map should be used

Operations complexity

	vector	list	deque	stack	queue	priority_queue	map	multimap	set	multiset
access ([])	O(1)	NA	O(1)	NA	NA	NA	O(log(n))	NA	NA	NA
list operations	O(n)	O(1)	O(n)	NA	NA	NA	O(log(n))	O(log(n))	O(log(n))	O(log(n))
front operations	NA	O(1)	O(1)	NA	O(1)	O(log(n))	NA	NA	NA	NA
back operations	O(1)	O(1)	O(1)	O(1)	O(1)	O(log(n))	NA	NA	NA	NA

Sequences

- Variable sized container with elements arranged in a linear order.
- Support insertion and removal of elements.
- Available operations depends on the type of sequence: e.g. vector allows random access by operator [], but not list.
- In the standard library: vector, list and deque.

Vector

- Defined in header <vector> in the namespace std
- Representation is most likely an array (representation is implementation dependent)
- A vector is suited for adding elements at the back with push_back()
- and for accessing elements through [] (unchecked) or at() (checked)
- It also allows list operations such as insert() or remove() but they are slow (O(n))

List

- Defined in header <list> in namespace std
- Representation is most likely a linked list (double)
- A list is optimized for insertion and deletion of elements
- No subscripting ([]) provided for list (contrary to vector)
- Additional methods for list manipulation
  - splice(): move elements from one list to another
  - sort(): sort elements of a list
  - merge(): merge two sorted lists

Deque

- Defined in header <deque> in namespace std
- A sequence optimized so that operations at both ends (back and front) are fast (as for list)
- Subscripting ([], at()) is also fast (as for vector)
- Other list operations for inserting and removing elements in the container are slow as for a vector
- A deque can be considered as a vector with the addition of methods for fast insert / remove at the beginning (push_front() and pop_front())
- Use a deque rather than a vector if you need to access both extremities

Adapters

- Adapters are containers constructed from other containers
- An adapter provide a restricted interface to an existing container
- Adapters are manipulated only through their specialized interfaces
- Adapters do not provide iterators like other containers (instead they are manipulated through their interface)
- In the standard libraries: stack, queue and priority_queue

Stack

- Defined in the header <stack> in namespace std
- A stack is providing mainly three operations: push, pop and top
- The stack in the standard library is defined as an interface to a container passed as a template argument
- By default a deque is used as a container
- Any container providing: back(), push_back() and pop_back() can be used
- Elements are added by push(), retrieved by top() and removed by pop()
- A stack can underflow: a call to pop() on an empty stack is undefined

Queue

- Defined in <queue> in namespace std
- Provide methods to access elements in the beginning and at the end and methods to add elements at the end and remove elements from the beginning
- Methods for manipulating a queue: back(), front(), push(), pop()
- Interface to a container providing the methods back(), front(), push_back() and pop_front()
- By default a deque is used
- Note that a vector can not be used as a container since it has no front() and pop_front() methods

Priority queue

- Defined in <queue> in namespace std
- A priority queue is a queue where each element is assigned a priority that controls which element goes to the top
- Interface: push() an element at the end, top() retrieve an element with highest priority and pop() the element with the highest priority
- By default comparison is using '<' and top() returns the element with highest priority. This behavior can be modified.
- A priority queue uses any container that provides front(), push_back() and pop_back(). By default a vector is used.
- push() and pop() require to keep elements in some order in the container. Time complexity: O(log(n))
- Usually priority queue are implemented using a heap (but this is implementation dependent)

Example

```
#include <queue>
struct Msg {
 int priority;
 bool operator< (const Msg& x) { return priority < x.priority; }
 // ...
};

int main(void) {
  std::priority_queue(Msg) pq;
  Msg m1(1, ...);
  Msg m2(2, ...);
  pq.push(m1); pq.push(m2);
  std::cout << pq.top() << std::endl; // get element with highest priority
  while (!pq.empty()) pq.pop();
}
```

Associative containers

- Associative containers keep pairs (key, value)
- key is used to retrieve the pair (key, value) in the collection
- An associative container can be thought of an array where a key is used instead of an index to retrieve a value.
- Associative containers in the standard library are: map, set, multimap and multiset.
- A map store pairs (key, value) with sorted (unique) keys
- A multimap is same but does not enforce the keys to be unique
- A set is a degenerate map where the key is also the value
- And a multiset is same as a multimap for a set

Map

- Defined in <map> in namespace std
- A map is a sequence of pairs (key, value)
- A map provides fast retrieval of pair based on a key
- At most one value is held for a key (so each key is unique)
- Keys stored in a map are sorted. Iteration through the pairs will be in sorted order of the keys.
- To allow fast retrieval, the representation for a map is usually a balanced search tree

Map iterators

- Usual methods returning iterators are provided: begin(), rbegin(), end(), rend()
- Iteration over a sequence of pair<K, T> where K is the key type and T the mapped value type
- Access to elements of a pair is done with first and second

```
void f(map<string, number>& phone_book) {
  typedef map<string,number>::iterator I;
  for (I p = phone_book.begin(); p != phone_book.end(); p++)
     cout << p->first << " - " << p->second;
}
```

- Keys are ordered so elements will appear in lexicographical order

Map subscripting

- Associative lookup (getting a value corresponding to a key) is provided by the subscript ([]) operator.
- If the key is not found, a pair (key, default value) is inserted in the map.

Ex:

```
map <string, int> m;
cout << m["a"] << endl; // print 0, default value for int
m["a"] = 5; // update the pair ("a",0) to ("a",5)
m["b"] = 1; // create new entry for b, init value to 0 then set to 1
```

- Lookup in a map has a cost of O(log(n)).

Map comparisons

- Insertion in a map requires key comparison to keep the order.
- Iteration through a map is done in order so it requires comparison as well.
- Lookup in a map given a key also requires to compare keys.
- By default the comparison used for keys is '<'.
- It is possible to specify an alternative comparison operators:

```
map<string, int> m1; // use '<' defined for string
map<string, int, cmp> m2; // use cmp as comp. operator
where:
class cmp {
public:
  bool operator() (const string& s1, const string& s2) const {}
};
```

Map functions

- Map functions allow to retrieve information for a given key in a map.
- m.find(k) will return an iterator to the element in m with key k.
- If the key is not found it returns m.end().
- Note that this behavior is different than for the subscript operator.

```
map<string, int>::iterator p = m.find("a");
if (p != m.end()) { /* "a" was found */ }
```

List operations for map

- m.insert(p) inserts the pair (key,value) in the map m.
- The pair is inserted only if there is not already a pair with that key.
- It returns a pair<iterator,bool>. The bool is true if the pair was successfully inserted in the map.
- There are other variants of insert().
- m.erase(it) erases the element indicated by iterator it. It returns the number of elements erased (can be > 1 for multimap and multiset)
  Ex:
  m.erase(m.find("a"));

Multimap

- A multimap is like a map but allows duplicate keys.
- Subscript operator is not provided, insertion is done with:
- iterator insert (const value_type&)

Example:

```
m.insert(make_pair("x",4));
m.insert(make_pair("x",5));
```

- Other methods for insertion are still available:
  lower_bound(), upper_bound(), etc

Set

- A set is like a map except that keys are also used for the values.
- Subscript operator ([]) is not available.
- Most of the other methods from map are available.

Multiset

- A multiset is a set but allows for duplicate keys.
- Contrary to a set, insert() returns an iterator and not a pair (since a key is not unique).
- Lookup operations are done with lower_bound() and upper_bound() operations (allow to access multiple occurrences of a key)

Other containers

- string, valarray and bitset also hold elements like containers.
- string:

provide subscripting, random access but is limited to hold characters.

- valarray:

is a vector optimized for numeric computation.

- bitset:

is an array of n bits. It is tailored for manipulating arrays of bits