**Vector**

**std::vector::vector:** Constructs a [vector](http://www.cplusplus.com/vector), initializing its contents depending on the constructor version used.

// constructors used in the same order as described above:

std::vector<int> first; // empty vector of ints

std::vector<int> second (4,100); // four ints with value 100

std::vector<int> third (second.begin(),second.end()); // iterating through second

std::vector<int> fourth (third); // a copy of third

// the iterator constructor can also be used to construct from arrays:

int myints[] = {16,2,77,29};

std::vector<int> fifth (myints, myints + sizeof(myints) / sizeof(int) );

**vector::assign:** Assigns new contents to the [vector](http://www.cplusplus.com/vector), replacing its current contents, and modifying its [size](http://www.cplusplus.com/vector::size) accordingly.

std::vector<int> first;

std::vector<int> second;

first.assign (7,100); // 7 ints with a value of 100

int myints[] = {1776,7,4};

std::vector<int>::iterator it;

it=first.begin()+1;

second.assign (it,first.end()-1); // the 5 central values of first

int myints[] = {1776,7,4};

third.assign (myints,myints+3); // assigning from array.

**vector::back:** Returns a reference to the last element in the [vector](http://www.cplusplus.com/vector). Unlike member [vector::end](http://www.cplusplus.com/vector::end), which returns an iterator just past this element, this function returns a direct reference. Calling this function on an [empty](http://www.cplusplus.com/vector::empty) container causes undefined behavior.

myvector.push\_back(10);

while (myvector.back() != 0)

{

myvector.push\_back ( myvector.back() -1 );

}

**vector::begin:** Returns an iterator pointing to the first element in the [vector](http://www.cplusplus.com/vector). Notice that, unlike member [vector::front](http://www.cplusplus.com/vector::front), which returns a reference to the first element, this function returns a [random access iterator](http://www.cplusplus.com/RandomAccessIterator) pointing to it. If the container is [empty](http://www.cplusplus.com/vector::empty), the returned iterator value shall not be dereferenced.

**vector::end:** Returns an iterator referring to the *past-the-end* element in the [vector](http://www.cplusplus.com/vector) container.  
The past-the-end element is the theoretical element that would follow the last element in the [vector](http://www.cplusplus.com/vector). It does not point to any element, and thus shall not be dereferenced.  
  
Because the ranges used by functions of the standard library do not include the element pointed by their closing iterator, this function is often used in combination with [vector::begin](http://www.cplusplus.com/vector::begin) to specify a range including all the elements in the container.If the container is [empty](http://www.cplusplus.com/vector::empty), this function returns the same as [vector::begin](http://www.cplusplus.com/vector::begin).

for (std::vector<int>::iterator it = myvector.begin() ; it != myvector.end(); ++it)

std::cout << ' ' << \*it;

**vector::capacity :** Returns the size of the storage space currently allocated for the [vector](http://www.cplusplus.com/vector), expressed in terms of elements. This *capacity* is not necessarily equal to the [vector size](http://www.cplusplus.com/vector::size). It can be equal or greater, with the extra space allowing to accommodate for growth without the need to reallocate on each insertion.  
  
Notice that this *capacity* does not suppose a limit on the size of the [vector](http://www.cplusplus.com/vector). When this *capacity* is exhausted and more is needed, it is automatically expanded by the container (reallocating it storage space). The theoretical limit on the [size](http://www.cplusplus.com/vector::size) of a [vector](http://www.cplusplus.com/vector) is given by member [max\_size](http://www.cplusplus.com/vector::max_size). The *capacity* of a [vector](http://www.cplusplus.com/vector) can be explicitly altered by calling member [vector::reserve](http://www.cplusplus.com/vector::reserve).

myvector.capacity();

**vector::cbegin:** Return const\_iterator to beginning. Returns a const\_iterator pointing to the first element in the container. A const\_iterator is an iterator that points to const content. This iterator can be increased and decreased (unless it is itself also const), just like the iterator returned by [vector::begin](http://www.cplusplus.com/vector::begin), but it cannot be used to modify the contents it points to, even if the [vector](http://www.cplusplus.com/vector) object is not itself const. If the container is [empty](http://www.cplusplus.com/vector::empty), the returned iterator value shall not be dereferenced.

**vector::cend**: Returns a const\_iterator pointing to the past-the-end element in the container. A const\_iterator is an iterator that points to const content. This iterator can be increased and decreased (unless it is itself also const), just like the iterator returned by [vector::end](http://www.cplusplus.com/vector::end), but it cannot be used to modify the contents it points to, even if the [vector](http://www.cplusplus.com/vector) object is not itself const. If the container is [empty](http://www.cplusplus.com/vector::empty), this function returns the same as [vector::cbegin](http://www.cplusplus.com/vector::cbegin). The value returned shall not be dereferenced.

for (auto it = myvector.cbegin(); it != myvector.cend(); ++it)

std::cout << ' ' << \*it;

**vector::clear:** Clear contentRemoves all elements from the vector (which are destroyed), leaving the container with a size of 0.A reallocation is not guaranteed to happen, and the vector capacity is not guaranteed to change due to calling this function. A typical alternative that forces a reallocation is to use swap:

myvector.clear();

**vector::crbegin:** Returns a const\_reverse\_iterator pointing to the last element in the container (i.e., its *reverse beginning*).

**vector::crend:** Returns a const\_reverse\_iterator pointing to the theoretical element preceding the first element in the container (which is considered its *reverse end*).

for (auto rit = myvector.crbegin(); rit != myvector.crend(); ++rit)

std::cout << ' ' << \*rit;

**vector::data:** Returns a direct pointer to the memory array used internally by the [vector](http://www.cplusplus.com/vector) to store its owned elements. Because elements in the [vector](http://www.cplusplus.com/vector) are guaranteed to be stored in contiguous storage locations in the same order as represented by the [vector](http://www.cplusplus.com/vector), the pointer retrieved can be offset to access any element in the array.

std::vector<int> myvector (5);

int\* p = myvector.data();

\*p = 10;

++p;

\*p = 20;

p[2] = 100;

std::cout << "myvector contains:";

for (unsigned i=0; i<myvector.size(); ++i)

std::cout << ' ' << myvector[i];

**vector::emplace:** Construct and insert element. The container is extended by inserting a new element at position. This new element is constructed in place using args as the arguments for its construction.  
This effectively increases the container [size](http://www.cplusplus.com/vector::size) by one.  
  
An automatic reallocation of the allocated storage space happens if -and only if- the new vector [size](http://www.cplusplus.com/vector::size) surpasses the current vector [capacity](http://www.cplusplus.com/vector::capacity).  
  
Because vectors use an array as their underlying storage, inserting elements in positions other than the [vector end](http://www.cplusplus.com/vector::end)causes the container to shift all the elements that were after position by one to their new positions. This is generally an inefficient operation compared to the one performed by other kinds of sequence containers (such as [list](http://www.cplusplus.com/list) or [forward\_list](http://www.cplusplus.com/forward_list)). See [emplace\_back](http://www.cplusplus.com/vector::emplace_back) for a member function that extends the container directly at the [end](http://www.cplusplus.com/vector::end).

std::vector<int> myvector = {10,20,30};

auto it = myvector.emplace ( myvector.begin()+1, 100 );

myvector.emplace ( it, 200 );

myvector.emplace ( myvector.end(), 300 );

std::cout << "myvector contains:";

for (auto& x: myvector)

std::cout << ' ' << x;

**vector::emplace\_back**: Construct and insert element at the end. Inserts a new element at the end of the [vector](http://www.cplusplus.com/vector), right after its current last element. This new element is constructed in place using args as the arguments for its constructor.  
  
This effectively increases the container [size](http://www.cplusplus.com/vector::size) by one, which causes an automatic reallocation of the allocated storage space if -and only if- the new vector [size](http://www.cplusplus.com/vector::size) surpasses the current vector [capacity](http://www.cplusplus.com/vector::capacity).  
  
The element is constructed in-place by calling [allocator\_traits::construct](http://www.cplusplus.com/allocator_traits::construct) with args forwarded.

std::vector<int> myvector = {10,20,30};

myvector.emplace\_back (100);

myvector.emplace\_back (200);

std::cout << "myvector contains:";

for (auto& x: myvector)

std::cout << ' ' << x;

**vector::empty** Returns whether the [vector](http://www.cplusplus.com/vector) is empty (i.e. whether its [size](http://www.cplusplus.com/vector::size) is 0).  
  
This function does not modify the container in any way. To clear the content of a [vector](http://www.cplusplus.com/vector), see [vector::clear](http://www.cplusplus.com/vector::clear).

while (!myvector.empty())

{

sum += myvector.back();

myvector.pop\_back();

}

**vector::erase:** Removes from the [vector](http://www.cplusplus.com/vector) either a single element (*position*) or a range of elements ([first,last)).  
  
This effectively reduces the container [size](http://www.cplusplus.com/vector::size) by the number of elements removed, which are destroyed.  
  
Because vectors use an array as their underlying storage, erasing elements in positions other than the [vector end](http://www.cplusplus.com/vector::end) causes the container to relocate all the elements after the segment erased to their new positions. This is generally an inefficient operation compared to the one performed for the same operation by other kinds of sequence containers (such as [list](http://www.cplusplus.com/list) or [forward\_list](http://www.cplusplus.com/forward_list)).

// erase the 6th element

myvector.erase (myvector.begin()+5);

// erase the first 3 elements:

myvector.erase (myvector.begin(),myvector.begin()+3);

**vector::front:** Access first element. Returns a reference to the first element in the [vector](http://www.cplusplus.com/vector).  
Unlike member [vector::begin](http://www.cplusplus.com/vector::begin), which returns an iterator to this same element, this function returns a direct reference. Calling this function on an [empty](http://www.cplusplus.com/vector::empty) container causes undefined behavior.

myvector.push\_back(78);

myvector.push\_back(16);

// now front equals 78, and back 16

myvector.front() -= myvector.back();

**vector::get\_allocator:** Returns a copy of the allocator object associated with the [vector](http://www.cplusplus.com/vector).

std::vector<int> myvector;

int \* p;

unsigned int i;

// allocate an array with space for 5 elements using vector's allocator:

p = myvector.get\_allocator().allocate(5);

// construct values in-place on the array:

for (i=0; i<5; i++) myvector.get\_allocator().construct(&p[i],i);

std::cout << "The allocated array contains:";

for (i=0; i<5; i++) std::cout << ' ' << p[i];

std::cout << '\n';

// destroy and deallocate:

for (i=0; i<5; i++) myvector.get\_allocator().destroy(&p[i]);

myvector.get\_allocator().deallocate(p,5);

**vector::insert:** The [vector](http://www.cplusplus.com/vector) is extended by inserting new elements before the element at the specified *position*, effectively increasing the container [size](http://www.cplusplus.com/vector::size) by the number of elements inserted.  
  
This causes an automatic reallocation of the allocated storage space if -and only if- the new vector [size](http://www.cplusplus.com/vector::size) surpasses the current vector [capacity](http://www.cplusplus.com/vector::capacity).  
  
Because vectors use an array as their underlying storage, inserting elements in positions other than the [vector end](http://www.cplusplus.com/vector::end)causes the container to relocate all the elements that were after *position* to their new positions. This is generally an inefficient operation compared to the one performed for the same operation by other kinds of sequence containers (such as [list](http://www.cplusplus.com/list) or [forward\_list](http://www.cplusplus.com/forward_list)).  
  
The parameters determine how many elements are inserted and to which values they are initialized:

std::vector<int> myvector (3,100);

std::vector<int>::iterator it;

it = myvector.begin();

it = myvector.insert ( it , 200 );

myvector.insert (it,2,300);

// "it" no longer valid, get a new one:

it = myvector.begin();

std::vector<int> anothervector (2,400);

myvector.insert (it+2,anothervector.begin(),anothervector.end());

int myarray [] = { 501,502,503 };

myvector.insert (myvector.begin(), myarray, myarray+3);

**vector::operator= :** Assigns new contents to the container, replacing its current contents, and modifying its [size](http://www.cplusplus.com/vector::size) accordingly.

std::vector<int> foo (3,0);

std::vector<int> bar (5,0);

bar = foo;

foo = std::vector<int>();

**vector::pop\_back:** Removes the last element in the [vector](http://www.cplusplus.com/vector), effectively reducing the container [size](http://www.cplusplus.com/vector::size) by one.

**vector::push\_back:** Add element at the end. Adds a new element at the end of the [vector](http://www.cplusplus.com/vector), after its current last element. The content of val is copied (or moved) to the new element.  
  
This effectively increases the container [size](http://www.cplusplus.com/vector::size) by one, which causes an automatic reallocation of the allocated storage space if -and only if- the new vector [size](http://www.cplusplus.com/vector::size) surpasses the current vector [capacity](http://www.cplusplus.com/vector::capacity).

myvector.push\_back (100);

myvector.push\_back (200);

myvector.push\_back (300);

while (!myvector.empty())

{

sum+=myvector.back();

myvector.pop\_back();

}

**vector::rbegin: Return reverse iterator to reverse beginning**

Returns a reverse iterator pointing to the last element in the [vector](http://www.cplusplus.com/vector) (i.e., its reverse beginning).  
*Reverse iterators* iterate backwards: increasing them moves them towards the beginning of the container. rbegin points to the element right before the one that would be pointed to by member [end](http://www.cplusplus.com/vector::end).  
Notice that unlike member [vector::back](http://www.cplusplus.com/vector::back), which returns a reference to this same element, this function returns a reverse [random access iterator](http://www.cplusplus.com/RandomAccessIterator).

**vector::rend:** Returns a *reverse iterator* pointing to the theoretical element preceding the first element in the [vector](http://www.cplusplus.com/vector) (which is considered its *reverse end*). The range between [vector::rbegin](http://www.cplusplus.com/vector::rbegin) and vector::rend contains all the elements of the [vector](http://www.cplusplus.com/vector) (in reverse order).

std::vector<int>::reverse\_iterator rit = myvector.rbegin();

for (; rit!= myvector.rend(); ++rit)

\*rit = ++i;

**vector::reserve:** Requests that the [vector capacity](http://www.cplusplus.com/vector::capacity) be at least enough to contain *n* elements.  
If *n* is greater than the current [vector capacity](http://www.cplusplus.com/vector::capacity), the function causes the container to reallocate its storage increasing its [capacity](http://www.cplusplus.com/vector::capacity) to *n* (or greater).  
  
In all other cases, the function call does not cause a reallocation and the [vector capacity](http://www.cplusplus.com/vector::capacity) is not affected.This function has no effect on the [vector size](http://www.cplusplus.com/vector::size) and cannot alter its elements.

bar.reserve(100); // capacity changed: 100

**vector::size:** Returns the number of elements in the [vector](http://www.cplusplus.com/vector). This is the number of actual objects held in the [vector](http://www.cplusplus.com/vector), which is not necessarily equal to its storage [capacity](http://www.cplusplus.com/vector::capacity).

myints.size()

**vector::swap: Swap content.** Exchanges the content of the container by the content of *x*, which is another [vector](http://www.cplusplus.com/vector) object of the same type. Sizes may differ. After the call to this member function, the elements in this container are those which were in *x* before the call, and the elements of *x* are those which were in this. All iterators, references and pointers remain valid for the swapped objects.  
Notice that a non-member function exists with the same name, [swap](http://www.cplusplus.com/vector:swap), overloading that algorithm with an optimization that behaves like this member function.

std::vector<int> foo (3,100); // three ints with a value of 100

std::vector<int> bar (5,200); // five ints with a value of 200

foo.swap(bar);