C++
Information
Tutorials
Reference
Articles
Forum

Reference
C library:
Containers:
<array>
<deque>
<forward_list>
<list>
<map>
<queue>
<set>
<stack>
<unordered_map>
<unordered_set>
<vector>
Input/Output:
Multi-threading:
Other:

<vector>
vector
vector<bool>

vector
vector::vector
vector::~vector
member functions:
vector::assign
vector::at
vector::back
vector::begin
vector::capacity
vector::cbegin
vector::cend
vector::clear
vector::crbegin

public member function

std::vector::emplace

<vector>

```
template <class... Args>
iterator emplace (const_iterator position, Args&&... args);
```

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 by one.

An automatic reallocation of the allocated storage space happens if -and only if- the new vector size surpasses the current vector capacity.

Because vectors use an array as their underlying storage, inserting elements in positions other than the 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 or forward_list). See emplace_back for a member function that extends the container directly at the end.

The element is constructed in-place by calling allocator_traits::construct with *args* forwarded.

A similar member function exists, insert, which either copies or moves existing objects into the container.

Parameters

`position`
Position in the container where the new element is inserted.
Member type `const_iterator` is a random access iterator type that points to a const element.

`args`
Arguments forwarded to construct the new element.

Return value

An iterator that points to the newly emplaced element.

Member type `iterator` is a random access iterator type that points to an element.

If a reallocation happens, the storage is allocated using the container's allocator, which may throw exceptions on failure (for the default allocator, `bad_alloc` is thrown if the allocation request does not succeed).

Example

```
// vector::emplace
#include <iostream>
#include <vector>

int main ()
{
  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;
  std::cout << '\n';

  return 0;
}
```

Edit & Run

Output:

```
myvector contains: 10 200 100 20 30 300
```

Complexity

Linear on the number of elements after *position* (moving).

If a reallocation happens, the reallocation is itself up to linear in the entire size.

Iterator validity

If a reallocation happens, all iterators, pointers and references related to this container are invalidated.
Otherwise, only those pointing to *position* and beyond are invalidated, with all iterators, pointers and references to elements before *position* guaranteed to keep referring to the same elements they were referring to before the call.

Data races

The container is modified.
If a reallocation happens, all contained elements are modified.
Otherwise, none of the elements before *position* is accessed, and concurrently accessing or modifying them is safe.

Exception safety

If *position* is end, and no reallocations happen, there are no changes in the container in case of exception (strong guarantee).
If a reallocation happens, the strong guarantee is also given if the type of the elements is either *copyable* or *no-throw moveable*.
Otherwise, the container is guaranteed to end in a valid state (basic guarantee).
If allocator_traits::construct is not supported with the appropriate arguments, or if *position* is not valid, it causes *undefined behavior*.

See also

vector::emplace_back	Construct and insert element at the end (public member function)
vector::insert	Insert elements (public member function)
vector::erase	Erase elements (public member function)
vector::assign	Assign vector content (public member function)