
8.4. Assignments

container& *container*::**operator =** (const *container*& *c*)

- Copy assignment operator.
- Assigns all elements of *c*; that is, it replaces all existing elements with copies of the elements of *c*.
- The operator may call the assignment operator for elements that have been overwritten, the copy constructor for appended elements, and the destructor of the element type for removed elements.
- Provided by array, vector, deque, list, forward list, set, multiset, map, multimap, unordered set, unordered multiset, unordered map, unordered multimap, string.

container& **container::operator =** (*container*&& *c*)

- Move assignment operator.
- Moves all elements of *c* to `*this`; that is, it replaces all existing elements with the elements of *c*.
- After this call, `C` is valid but has an unspecified value.
- Available since C++11.
- Provided by vector, deque, list, forward list, set, multiset, map, multimap, unordered set, unordered multiset, unordered map, unordered multimap, string.

container& **container::operator =** (*initializer-list*)

- Assigns all elements of *initializer-list*; that is, it replaces all existing elements with copies of the passed elements.
- The operator may call the assignment operator for elements that have been overwritten, the copy constructor for appended elements, and the destructor of the element type for removed elements.
- Available since C++11.
- Provided by vector, deque, list, forward list, set, multiset, map, multimap, unordered set, unordered multiset, unordered map, unordered multimap, string.

void **container::assign** (*initializer-list*)

- Assigns all elements of the *initializer-list*; that is, it replaces all existing elements with copies of the passed elements.
- Available since C++11.
- Provided by vector, deque, list, forward list, string.

void **array::fill** (const T& *value*)

- Assigns *value* to all elements; that is, it replaces all existing elements with copies of the *value*.
- Available since C++11.
- Provided by array.

void **container::assign** (size_type *num*, const T& *value*)

- Assigns *num* occurrences of *value*; that is, it replaces all existing elements by *num* copies of *value*.
- `T` has to be the element type.
- Provided by vector, deque, list, forward list, string.

void **container::assign** (InputIterator *beg*, InputIterator *end*)

- Assigns all elements of the range `[` *beg* `,` *end* `)`; that is, it replaces all existing elements with copies of the elements of `[` *beg* `,` *end* `)`.
- This function is a member template (see Section 3.2, page 34). Thus, the elements of the source range may have any type convertible into the element type of the container.
- Provided by vector, deque, list, forward list, string.

void **container::swap** (*container*& *c*)
void **swap** (*container*& *c1*, *container*& *c2*)

- Swap the contents with *c* or between *c1* and *c2*, respectively.
- Both swap:

  - The container's elements
  - Their sorting criterion, equivalence predicate, and hash function object, if any. The references, pointers, and iterators referring to elements swap their containers, because they still refer to the same swapped elements afterward.

- Arrays can't internally just swap pointers. Thus, `swap()` has linear complexity, and iterators and references refer to the same container but different elements afterward.
- For associative containers, the function may throw only if copying or assigning the comparison criterion may throw. For unordered containers, the function may throw only if the equivalence predicate or the hash function object may throw. For all other containers, the function does not throw.
- Complexity: constant, in general. For arrays it is linear.
- Due to its complexity, you should always prefer `swap()` over a copy assignment when you no longer need the assigned object (see Section 7.1.2, page 258).
- Provided by array, vector, deque, list, forward list, set, multiset, map, multimap, unordered set, unordered multiset, unordered map, unordered multimap, string.