Stl Container Methods

Nathan Warner

Northern Illinois
University

Computer Science
Northern Illinois University
United States

Contents

Vectors

1.1 Nonmodifying Operations

- `c.empty()`
  Returns whether the container is empty (equivalent to `size()` == 0 but might be faster).
- `c.size()`
  Returns the current number of elements.
- `c.max_size()`
  Returns the maximum number of elements possible.
- `c.capacity()`
  Returns the maximum possible number of elements without reallocation.
- `c.reserve(num)`
  Enlarges capacity, if not enough yet[6].
- `c.shrink_to_fit()`
  Requests to reduce capacity to fit the number of elements (since C++11)[6].
- `c1 == c2`
  Returns whether `c1` is equal to `c2` (calls `==` for the elements).
- `c1 != c2`
  Returns whether `c1` is not equal to `c2` (equivalent to `!(c1 == c2)`).
- `c1 < c2`
  Returns whether `c1` is less than `c2`.
- `c1 > c2`
  Returns whether `c1` is greater than `c2` (equivalent to `c2 < c1`).
- `c1 <= c2`
  Returns whether `c1` is less than or equal to `c2` (equivalent to `!(c2 < c1)`).
- `c1 >= c2`
  Returns whether `c1` is greater than or equal to `c2` (equivalent to `!(c1 < c2)`).

1.2 Assignments

- `c = c2`
  Assigns all elements of `c2` to `c`.
- `c = rv`
  Move assigns all elements of the rvalue `rv` to `c` (since C++11).
- `c = initlist`
  Assigns all elements of the initializer list `initlist` to `c` (since C++11).
- `c.assign(n, elem)`
  Assigns `n` copies of element `elem`.
- `c.assign(beg, end)`
  Assigns the elements of the range [`beg`, `end`].
- `c.assign(initlist)`
  Assigns all the elements of the initializer list `initlist`.
- `c1.swap(c2)`
  Swaps the data of `c1` and `c2`.
- `swap(c1, c2)`
  Swaps the data of `c1` and `c2`.

1.3 Element access

- `c[idx]`
  Returns the element with index `idx` *(no range checking)*.
- `c.at(idx)`
  Returns the element with index `idx` *(throws range-error exception if `idx` is out of range)*.
- `c.front()`
  Returns the first element *(no check whether a first element exists)*.
- `c.back()`
  Returns the last element *(no check whether a last element exists)*.

1.4 Inserting and Removing Elements

- `c.push_back(elem)`
  Appends a copy of `elem` at the end.

- `c.pop_back()`
  Removes the last element (does not return it).

- `c.insert(pos, elem)`
  Inserts a copy of `elem` before iterator position `pos` and returns the position of the new element.

- `c.insert(pos, n, elem)`
  Inserts `n` copies of `elem` before iterator position `pos` and returns the position of the first new element (or `pos` if there is no new element).

- `c.insert(pos, beg, end)`
  Inserts a copy of all elements of the range [`beg, end`] before iterator position `pos` and returns the position of the first new element (or `pos` if there is no new element).

- `c.insert(pos, initlist)`
  Inserts a copy of all elements of the initializer list `initlist` before iterator position `pos` and returns the position of the first new element (or `pos` if there is no new element; since C++11).

- `c.emplace(pos, args...)`
  Inserts a copy of an element initialized with `args` before iterator position `pos` and returns the position of the new element (since C++11).

- `c.emplace_back(args...)`
  Appends a copy of an element initialized with `args` at the end (returns nothing; since C++11).

- `c.erase(pos)`
  Removes the element at iterator position `pos` and returns the position of the next element.

- `c.erase(beg, end)`
  Removes all elements of the range [`beg, end`] and returns the position of the next element.

- `c.resize(num)`
  Changes the number of elements to `num` (if `size()` grows, new elements are created by their default constructor).

- `c.resize(num, elem)`
  Changes the number of elements to `num` (if `size()` grows, new elements are copies of `elem`).

- `c.clear()`
  Removes all elements (empties the container).

Deque

2.1 Nonmodifying Operations

- `c.empty()`
  Returns whether the container is empty (equivalent to `size()` == 0 but might be faster).
- `c.size()`
  Returns the current number of elements.
- `c.max_size()`
  Returns the maximum number of elements possible.
- `c.shrink_to_fit()`
  Requests to reduce capacity to fit the number of elements (since C++11)[6].
- `c1 == c2`
  Returns whether `c1` is equal to `c2` (calls `==` for the elements).
- `c1 != c2`
  Returns whether `c1` is not equal to `c2` (equivalent to `!(c1 == c2)`).
- `c1 < c2`
  Returns whether `c1` is less than `c2`.
- `c1 > c2`
  Returns whether `c1` is greater than `c2` (equivalent to `c2 < c1`).
- `c1 <= c2`
  Returns whether `c1` is less than or equal to `c2` (equivalent to `!(c2 < c1)`).
- `c1 >= c2`
  Returns whether `c1` is greater than or equal to `c2` (equivalent to `!(c1 < c2)`).

2.2 Assignments

- `c = c2`
  Assigns all elements of `c2` to `c`.
- `c = rv`
  Move assigns all elements of the rvalue `rv` to `c` (since C++11).
- `c = initlist`
  Assigns all elements of the initializer list `initlist` to `c` (since C++11).
- `c.assign(n, elem)`
  Assigns `n` copies of element `elem`.
- `c.assign(beg, end)`
  Assigns the elements of the range [`beg`, `end`].
- `c.assign(initlist)`
  Assigns all the elements of the initializer list `initlist`.
- `c1.swap(c2)`
  Swaps the data of `c1` and `c2`.
- `swap(c1, c2)`
  Swaps the data of `c1` and `c2`.

2.3 Element access

- `c[idx]`
  Returns the element with index `idx` *(no range checking)*.
- `c.at(idx)`
  Returns the element with index `idx` *(throws range-error exception if `idx` is out of range)*.
- `c.front()`
  Returns the first element *(no check whether a first element exists)*.
- `c.back()`
  Returns the last element *(no check whether a last element exists)*.

2.4 Inserting and Removing Elements

- `c.push_back(elem)`
  Appends a copy of `elem` at the end.
- `c.pop_back()`
  Removes the last element (does not return it).
- `c.push_front()` Appends a copy of `elem` at the front.
- `c.pop_front()` Removes the first element (does not return it).
- `c.insert(pos, elem)`
  Inserts a copy of `elem` before iterator position `pos` and returns the position of the new element.
- `c.insert(pos, n, elem)`
  Inserts `n` copies of `elem` before iterator position `pos` and returns the position of the first new element (or `pos` if there is no new element).
- `c.insert(pos, beg, end)`
  Inserts a copy of all elements of the range [`beg, end`] before iterator position `pos` and returns the position of the first new element (or `pos` if there is no new element).
- `c.insert(pos, initlist)`
  Inserts a copy of all elements of the initializer list `initlist` before iterator position `pos` and returns the position of the first new element (or `pos` if there is no new element; since C++11).
- `c.emplace(pos, args...)`
  Inserts a copy of an element initialized with `args` before iterator position `pos` and returns the position of the new element (since C++11).
- `c.emplace_back(args...)`
  Appends a copy of an element initialized with `args` at the end (returns nothing; since C++11).
- `c.erase(pos)`
  Removes the element at iterator position `pos` and returns the position of the next element.
- `c.erase(beg, end)`
  Removes all elements of the range [`beg, end`] and returns the position of the next element.
- `c.resize(num)`
  Changes the number of elements to `num` (if `size()` grows, new elements are created by their default constructor).
- `c.resize(num, elem)`
  Changes the number of elements to `num` (if `size()` grows, new elements are copies of `elem`).
- `c.clear()`
  Removes all elements (empties the container).

Lists

3.1 Nonmod

- `c.empty()`
  Returns whether the container is empty (equivalent to `size()` == 0 but might be faster).
- `c.size()`
  Returns the current number of elements.
- `c.max_size()`
  Returns the maximum number of elements possible.
- `c1 == c2`
  Returns whether `c1` is equal to `c2` (calls `==` for the elements).
- `c1 != c2`
  Returns whether `c1` is not equal to `c2` (equivalent to `!(c1 == c2)`).
- `c1 < c2`
  Returns whether `c1` is less than `c2`.
- `c1 > c2`
  Returns whether `c1` is greater than `c2` (equivalent to `c2 < c1`).
- `c1 <= c2`
  Returns whether `c1` is less than or equal to `c2` (equivalent to `!(c2 < c1)`).
- `c1 >= c2`
  Returns whether `c1` is greater than or equal to `c2` (equivalent to `!(c1 < c2)`).

3.2 Assignment

- `c = c2`
  Assigns all elements of `c2` to `c`.
- `c = rv`
  Move assigns all elements of the rvalue `rv` to `c` (since C++11).
- `c = initlist`
  Assigns all elements of the initializer list `initlist` to `c` (since C++11).
- `c.assign(n, elem)`
  Assigns `n` copies of element `elem`.
- `c.assign(beg, end)`
  Assigns the elements of the range [`beg`, `end`].
- `c.assign(initlist)`
  Assigns all the elements of the initializer list `initlist`.
- `c1.swap(c2)`
  Swaps the data of `c1` and `c2`.
- `swap(c1, c2)`
  Swaps the data of `c1` and `c2`.

3.3 Element access

- `c.front()`: No check whether the element exists
- `c.back()`: No check whether the element exists

3.4 Insert and Remove

- `c.push_back(elem)`
  Appends a copy of `elem` at the end.
- `c.pop_back()`
  Removes the last element (does not return it).
- `c.push_front(elem)`
  Inserts a copy of `elem` at the beginning.
- `c.pop_front()`
  Removes the first element (does not return it).
- `c.insert(pos, elem)`
  Inserts a copy of `elem` before iterator position `pos` and returns the position of the new element.
- `c.insert(pos, n, elem)`
  Inserts `n` copies of `elem` before iterator position `pos` and returns the position of the first new element (or `pos` if there is no new element).
- `c.insert(pos, beg, end)`
  Inserts a copy of all elements of the range `[beg, end]` before iterator position `pos` and returns the position of the first new element (or `pos` if there is no new element).
- `c.insert(pos, inilist)`
  Inserts a copy of all elements of the initializer list `inilist` before iterator position `pos` and returns the position of the first new element (or `pos` if there is no new element; since C++11).
- `c.emplace(pos, args...)`
  Inserts a copy of an element initialized with `args` before iterator position `pos` and returns the position of the new element (since C++11).
- `c.emplace_back(args...)`
  Appends a copy of an element initialized with `args` at the end (returns nothing; since C++11).
- `c.emplace_front(args...)`
  Inserts a copy of an element initialized with `args` at the beginning (returns nothing; since C++11).
- `c.erase(pos)`
  Removes the element at iterator position `pos` and returns the position of the next element.
- `c.erase(beg, end)`
  Removes all elements of the range `[beg, end]` and returns the position of the next element.
- `c.remove(val)`
  Removes all elements with value `val`.
- `c.remove_if(op)`
  Removes all elements for which `op(elem)` yields `true`.
- `c.resize(num)`
  Changes the number of elements to `num` (if `size()` grows, new elements are created by their default constructor).

- `c.resize(num, elem)`
  Changes the number of elements to `num` (if `size()` grows, new elements are copies of `elem`).
- `c.clear()`
  Removes all elements (empties the container).

3.5 Special Modifying Operations for Lists

- `c.unique()`
  Removes duplicates of consecutive elements with the same value.
- `c.unique(op)`
  Removes duplicates of consecutive elements, for which `op()` yields `true`.
- `c.splice(pos, c2)`
  Moves all elements of `c2` to `c` in front of the iterator position `pos`.
- `c.splice(pos, c2, c2pos)`
  Moves the element at `c2pos` in `c2` in front of `pos` of list `c` (`c` and `c2` may be identical).
- `c.splice(pos, c2, c2beg, c2end)`
  Moves all elements of the range `[c2beg, c2end)` in `c2` in front of `pos` of list `c` (`c` and `c2` may be identical).
- `c.sort()`
  Sorts all elements with operator `<`.
- `c.sort(op)`
  Sorts all elements with `op()`.
- `c.merge(c2)`
  Assuming that both containers contain the elements sorted, moves all elements of `c2` into `c` so that all elements are merged and still sorted.
- `c.merge(c2, op)`
  Assuming that both containers contain the elements sorted due to the sorting criterion `op()`, moves all elements of `c2` into `c` so that all elements are merged and still sorted according to `op()`.
- `c.reverse()`
  Reverses the order of all elements.

3.6 Sorting

- `c.sort()`: Sorts the list

Forward_list

4.1 Non Modifying Operations

- `c.empty()` Returns whether the container is empty
- `c.max_size()` Returns the maximum number of elements possible
- `c1 == c2` Returns whether `c1` is equal to `c2` (calls `==` for the elements)
- `c1 != c2` Returns whether `c1` is not equal to `c2` (equivalent to `!(c1==c2)`)
- `c1 < c2` Returns whether `c1` is less than `c2`
- `c1 > c2` Returns whether `c1` is greater than `c2` (equivalent to `c2<c1`)
- `c1 <= c2` Returns whether `c1` is less than or equal to `c2` (equivalent to `!(c2<c1)`)
- `c1 >= c2` Returns whether `c1` is greater than or equal to `c2` (equivalent to `!(c1<c2)`)

4.2 Assignments

- `c = c2` Assigns all elements of `c2` to `c`
- `c = rv` Move assigns all elements of the rvalue `rv` to `c` (since C++11)
- `c = initlist` Assigns all elements of the initializer list `initlist` to `c` (since C++11)
- `c.assign(n,elem)` Assigns `n` copies of element `elem`
- `c.assign(beg,end)` Assigns the elements of the range `[beg,end)`
- `c.assign(initlist)` Assigns all the elements of the initializer list `initlist`
- `c1.swap(c2)` Swaps the data of `c1` and `c2`
- `swap(c1,c2)` Swaps the data of `c1` and `c2`

4.3 Element access

- **c.front()**: Returns the first element (no check whether a first element exists)

4.4 Iterator functions

- `c.begin()` Returns a bidirectional iterator for the first element
- `c.end()` Returns a bidirectional iterator for the position after the last element
- `c.cbegin()` Returns a constant bidirectional iterator for the first element (since C++11)
- `c.cend()` Returns a constant bidirectional iterator for the position after the last element (since C++11)
- `c.before_begin()` Returns a forward iterator for the position before the first element
- `c.cbefore_begin()` Returns a constant forward iterator for the position before the first element

4.5 Insert and Remove

- `c.push_front(elem)` Inserts a copy of `elem` at the beginning
- `c.pop_front()` Removes the first element (does not return it)
- `c.insert_after(pos,elem)` Inserts a copy of `elem` after iterator position `pos` and returns the position of the new element
- `c.insert_after(pos,n,elem)` Inserts `n` copies of `elem` after iterator position `pos` and returns the position of the first new element (or `pos` if there is no new element)
- `c.insert_after(pos,beg,end)` Inserts a copy of all elements of the range `[beg,end)` after iterator position `pos` and returns the position of the first new element (or `pos` if there is no new element)
- `c.insert_after(pos,initlist)` Inserts a copy of all elements of the initializer list `initlist` after iterator position `pos` and returns the position of the first new element (or `pos` if there is no new element)
- `c.emplace_after(pos,args...)` Inserts a copy of an element initialized with `args` after iterator position `pos` and returns the position of the new element (since C++11)
- `c.emplace_front(args...)` Inserts a copy of an element initialized with `args` at the beginning (returns nothing; since C++11)
- `c.erase_after(pos)` Removes the element after iterator position `pos` (returns nothing)
- `c.erase_after(beg,end)` Removes all elements of the range `[beg,end)` (returns nothing)
- `c.remove(val)` Removes all elements with value `val`
- `c.remove_if(op)` Removes all elements for which `op(elem)` yields `true`
- `c.resize(num)` Changes the number of elements to `num` (if size grows, new elements are created by their default constructor)
- `c.resize(num,elem)` Changes the number of elements to `num` (if size grows, new elements are copies of `elem`)
- `c.clear()` Removes all elements (empties the container)

4.6 Special Modifying

- `c.unique()` Removes duplicates of consecutive elements with the same value
- `c.unique(op)` Removes duplicates of consecutive elements, for which `op()` yields true
- `c.splice_after(pos,c2)` Moves all elements of `c2` to `c` right behind the iterator position `pos`
- `c.splice_after(pos,c2,c2pos)` Moves the element behind `c2pos` in `c2` right after `pos` of forward list `c` (`c` and `c2` may be identical)
- `c.splice_after(pos,c2,c2beg,c2end)` Moves all elements between `c2beg` and `c2end` (both not included) in `c2` right after `pos` of forward list `c` (`c` and `c2` may be identical)
- `c.sort()` Sorts all elements with `operator<`

- `c.sort(op)` Sorts all elements with `op()`
- `c.merge(c2)` Assuming that both containers contain the elements sorted, moves all elements of `c2` into `c` so that all elements are merged and still sorted
- `c.merge(c2,op)` Assuming that both containers contain the elements sorted by the sorting criterion `op()`, moves all elements of `c2` into `c` so that all elements are merged and still sorted according to `op()`
- `c.reverse()` Reverses the order of all elements

Sets and multisets

5.1 Assignments

- `c = c2` Assigns all elements of `c2` to `c`
- `c = rv` Move assigns all elements of the rvalue `rv` to `c` (since C++11)
- `c = initlist` Assigns all elements of the initializer list `initlist` to `c` (since C++11)
- `c1.swap(c2)` Swaps the data of `c1` and `c2`
- `swap(c1,c2)` Swaps the data of `c1` and `c2`

5.2 Non Modifying

- `c.key_comp()` Returns the comparison criterion
- `c.value_comp()` Returns the comparison criterion for values as a whole (same as `key_comp()`)
- `c.empty()` Returns whether the container is empty (equivalent to `size()==0` but might be faster)
- `c.size()` Returns the current number of elements
- `c.max_size()` Returns the maximum number of elements possible
- `c1 == c2` Returns whether `c1` is equal to `c2` (calls `==` for the elements)
- `c1 != c2` Returns whether `c1` is not equal to `c2` (equivalent to `!(c1==c2)`)
- `c1 < c2` Returns whether `c1` is less than `c2`
- `c1 > c2` Returns whether `c1` is greater than `c2` (equivalent to `c2<c1`)
- `c1 <= c2` Returns whether `c1` is less than or equal to `c2` (equivalent to `!(c2<c1)`)
- `c1 >= c2` Returns whether `c1` is greater than or equal to `c2` (equivalent to `!(c1<c2)`)

5.3 Search operations

- `c.count(val)` Returns the number of elements with value `val`
- `c.find(val)` Returns the position of the first element with value `val` (or `end()` if none found)
- `c.lower_bound(val)` Returns the first position where `val` would get inserted (the first element `>= val`)
- `c.upper_bound(val)` Returns the last position where `val` would get inserted (the first element `> val`)
- `c.equal_range(val)` Returns a range with all elements with a value equal to `val` (i.e., the first and last position where `val` would get inserted)

5.4 Iterator functions

- `c.begin()` Returns a bidirectional iterator for the first element
- `c.end()` Returns a bidirectional iterator for the position after the last element
- `c.cbegin()` Returns a constant bidirectional iterator for the first element (since C++11)
- `c.cend()` Returns a constant bidirectional iterator for the position after the last element (since C++11)
- `c.rbegin()` Returns a reverse iterator for the first element of a reverse iteration
- `c.rend()` Returns a reverse iterator for the position after the last element of a reverse iteration
- `c.crbegin()` Returns a constant reverse iterator for the first element of a reverse iteration (since C++11)
- `c.crend()` Returns a constant reverse iterator for the position after the last element of a reverse iteration (since C++11)

5.5 Search and remove

- `c.insert(val)` Inserts a copy of `val` and returns the position of the new element and, for sets, whether it succeeded
- `c.insert(pos,val)` Inserts a copy of `val` and returns the position of the new element (`pos` is used as a hint pointing to where the insert should start the search)
- `c.insert(beg,end)` Inserts a copy of all elements of the range `[beg,end)` (returns nothing)
- `c.insert(initlist)` Inserts a copy of all elements in the initializer list `initlist` (returns nothing; since C++11)
- `c.emplace(args...)` Inserts a copy of an element initialized with `args` and returns the position of the new element and, for sets, whether it succeeded (since C++11)
- `c.emplace_hint(pos,args...)` Inserts a copy of an element initialized with `args` and returns the position of the new element (`pos` is used as a hint pointing to where the insert should start the search)
- `c.erase(val)` Removes all elements equal to `val` and returns the number of removed elements
- `c.erase(pos)` Removes the element at iterator position `pos` and returns the following position (returned nothing before C++11)
- `c.erase(beg,end)` Removes all elements of the range `[beg,end)` and returns the following position (returned nothing before C++11)
- `c.clear()` Removes all elements (empties the container)

Maps and multimaps

6.1 Assignments

- `c = c2` Assigns all elements of `c2` to `c`
- `c = rv` Move assigns all elements of the rvalue `rv` to `c` (since C++11)
- `c = initlist` Assigns all elements of the initializer list `initlist` to `c` (since C++11)
- `c1.swap(c2)` Swaps the data of `c1` and `c2`
- `swap(c1,c2)` Swaps the data of `c1` and `c2`

6.2 Non Modifying operations

- `c.key_comp()` Returns the comparison criterion
- `c.value_comp()` Returns the comparison criterion for values as a whole (an object that compares the key in a key/value pair)
- `c.empty()` Returns whether the container is empty (equivalent to `size()==0` but might be faster)
- `c.size()` Returns the current number of elements
- `c.max_size()` Returns the maximum number of elements possible
- `c1 == c2` Returns whether `c1` is equal to `c2` (calls `==` for the elements)
- `c1 != c2` Returns whether `c1` is not equal to `c2` (equivalent to `!(c1==c2)`)
- `c1 < c2` Returns whether `c1` is less than `c2`
- `c1 > c2` Returns whether `c1` is greater than `c2` (equivalent to `c2<c1`)
- `c1 <= c2` Returns whether `c1` is less than or equal to `c2` (equivalent to `!(c2<c1)`)
- `c1 >= c2` Returns whether `c1` is greater than or equal to `c2` (equivalent to `!(c1<c2)`)

6.3 Search operations

- `c.count(val)` Returns the number of elements with key `val`
- `c.find(val)` Returns the position of the first element with key `val` (or `end()` if none found)
- `c.lower_bound(val)` Returns the first position where an element with key `val` would get inserted (the first element with a key `>= val`)
- `c.upper_bound(val)` Returns the last position where an element with key `val` would get inserted (the first element with a key `> val`)
- `c.equal_range(val)` Returns a range with all elements with a key equal to `val` (i.e., the first and last positions where an element with key `val` would get inserted)

6.4 Iterator functions

- `c.begin()` Returns a bidirectional iterator for the first element
- `c.end()` Returns a bidirectional iterator for the position after the last element
- `c.cbegin()` Returns a constant bidirectional iterator for the first element (since C++11)
- `c.cend()` Returns a constant bidirectional iterator for the position after the last element (since C++11)
- `c.rbegin()` Returns a reverse iterator for the first element of a reverse iteration
- `c.rend()` Returns a reverse iterator for the position after the last element of a reverse iteration
- `c.crbegin()` Returns a constant reverse iterator for the first element of a reverse iteration (since C++11)
- `c.crend()` Returns a constant reverse iterator for the position after the last element of a reverse iteration (since C++11)

6.5 Insert and remove

- `c.insert(val)` Inserts a copy of `val` and returns the position of the new element and, for maps, whether it succeeded
- `c.insert(pos,val)` Inserts a copy of `val` and returns the position of the new element (`pos` is used as a hint pointing to where the insert should start the search)
- `c.insert(beg,end)` Inserts a copy of all elements of the range `[beg,end)` (returns nothing)
- `c.insert(initlist)` Inserts a copy of all elements in the initializer list `initlist` (returns nothing; since C++11)
- `c.emplace(args...)` Inserts a copy of an element initialized with `args` and returns the position of the new element and, for maps, whether it succeeded (since C++11)
- `c.emplace_hint(pos,args...)` Inserts a copy of an element initialized with `args` and returns the position of the new element (`pos` is used as a hint pointing to where the insert should start the search)
- `c.erase(val)` Removes all elements equal to `val` and returns the number of removed elements
- `c.erase(pos)` Removes the element at iterator position `pos` and returns the following position (returned nothing before C++11)
- `c.erase(beg,end)` Removes all elements of the range `[beg,end)` and returns the following position (returned nothing before C++11)
- `c.clear()` Removes all elements (empties the container)

6.6 Element access

- `c[key]` Inserts an element with key, if it does not yet exist, and returns a reference to the value of the element with key (only for nonconstant maps)
- `c.at(key)` Returns a reference to the value of the element with key (since C++11)