Stl Container Methods

Nathan Warner

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.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).

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.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