Non-modifying sequence operations

Defined in header <algorithm>

all_of (C++11) any_of (C++11) none_of (C++11)	checks if a predicate is true for all, any or none of the elements in a range (function template)
ranges::all_of (C++20) ranges::any_of (C++20) ranges::none_of (C++20)	checks if a predicate is true for all, any or none of the elements in a range (niebloid)
for_each	applies a function to a range of elements (function template)
ranges::for_each (C++20)	applies a function to a range of elements (niebloid)
for_each_n (C++17)	applies a function object to the first n elements of a sequence (function template)
ranges::for_each_n (C++20)	applies a function object to the first n elements of a sequence (niebloid)
count count_if	returns the number of elements satisfying specific criteria (function template)
ranges::count (C++20) ranges::count_if (C++20)	returns the number of elements satisfying specific criteria (niebloid)
mismatch	finds the first position where two ranges differ (function template)
ranges::mismatch (C++20)	finds the first position where two ranges differ (niebloid)
find find_if find_if_not (C++11)	finds the first element satisfying specific criteria (function template)
ranges::find (C++20) ranges::find_if (C++20) ranges::find_if_not (C++20)	finds the first element satisfying specific criteria (niebloid)
find_end	finds the last sequence of elements in a certain range (function template)
ranges::find_end (C++20)	finds the last sequence of elements in a certain range (niebloid)
find_first_of	searches for any one of a set of elements (function template)
ranges::find_first_of (C++20)	searches for any one of a set of elements (niebloid)
adjacent_find	finds the first two adjacent items that are equal (or satisfy a given predicate) (function template)
ranges::adjacent_find (C++20)	finds the first two adjacent items that are equal (or satisfy a given predicate) (niebloid)
search	searches for a range of elements (function template)
ranges::search (C++20)	searches for a range of elements (niebloid)
search_n	searches a range for a number of consecutive copies of an element (function template)
ranges::search_n (C++20)	searches for a number consecutive copies of an element in a range (niebloid)
ranges::starts_with (C++23)	checks whether a range starts with another range (niebloid)
ranges::ends_with (C++23)	checks whether a range ends with another range (niebloid)

Modifying sequence operations

Defined in header <algorithm>

copy copy_if (C++11)	copies a range of elements to a new location (function template)
ranges::copy (C++20) ranges::copy_if (C++20)	copies a range of elements to a new location (niebloid)
copy_n (C++11)	copies a number of elements to a new location (function template)
ranges::copy_n (C++20)	copies a number of elements to a new location (niebloid)
copy_backward	copies a range of elements in backwards order (function template)
ranges::copy_backward (C++20)	copies a range of elements in backwards order (niebloid)
move (C++11)	moves a range of elements to a new location (function template)
ranges::move (C++20)	moves a range of elements to a new location (niebloid)
move_backward (C++11)	moves a range of elements to a new location in backwards order (function template)
ranges::move_backward (C++20)	moves a range of elements to a new location in backwards order (niebloid)
fill	copy-assigns the given value to every element in a range (function template)
ranges::fill (C++20)	assigns a range of elements a certain value (niebloid)
fill_n	copy-assigns the given value to N elements in a range (function template)
ranges::fill_n (C++20)	assigns a value to a number of elements (niebloid)
transform	applies a function to a range of elements, storing results in a destination range (function template)
ranges::transform (C++20)	applies a function to a range of elements (niebloid)
generate	assigns the results of successive function calls to every element in a range (function template)
ranges::generate (C++20)	saves the result of a function in a range (niebloid)
generate_n	assigns the results of successive function calls to N elements in a range (function template)
ranges::generate_n (C++20)	saves the result of N applications of a function (niebloid)

`ranges::remove` (C++20) `ranges::remove_if` (C++20)	removes elements satisfying specific criteria (niebloid)
`remove_copy` `remove_copy_if`	copies a range of elements omitting those that satisfy specific criteria (function template)
`ranges::remove_copy` (C++20) `ranges::remove_copy_if` (C++20)	copies a range of elements omitting those that satisfy specific criteria (niebloid)
`replace` `replace_if`	replaces all values satisfying specific criteria with another value (function template)
`ranges::replace` (C++20) `ranges::replace_if` (C++20)	replaces all values satisfying specific criteria with another value (niebloid)
`replace_copy` `replace_copy_if`	copies a range, replacing elements satisfying specific criteria with another value (function template)
`ranges::replace_copy` (C++20) `ranges::replace_copy_if` (C++20)	copies a range, replacing elements satisfying specific criteria with another value (niebloid)
`swap`	swaps the values of two objects (function template)
`swap_ranges`	swaps two ranges of elements (function template)
`ranges::swap_ranges` (C++20)	swaps two ranges of elements (niebloid)
`iter_swap`	swaps the elements pointed to by two iterators (function template)
`reverse`	reverses the order of elements in a range (function template)
`ranges::reverse` (C++20)	reverses the order of elements in a range (niebloid)
`reverse_copy`	creates a copy of a range that is reversed (function template)
`ranges::reverse_copy` (C++20)	creates a copy of a range that is reversed (niebloid)
`rotate`	rotates the order of elements in a range (function template)
`ranges::rotate` (C++20)	rotates the order of elements in a range (niebloid)
`rotate_copy`	copies and rotate a range of elements (function template)
`ranges::rotate_copy` (C++20)	copies and rotate a range of elements (niebloid)
`shift_left` `shift_right` (C++20)	shifts elements in a range (function template)
`random_shuffle` (until C++17) `shuffle` (C++11)	randomly re-orders elements in a range (function template)
`ranges::shuffle` (C++20)	randomly re-orders elements in a range (niebloid)
`sample` (C++17)	selects n random elements from a sequence (function template)
`ranges::sample` (C++20)	selects n random elements from a sequence (niebloid)
`unique`	removes consecutive duplicate elements in a range (function template)
`ranges::unique` (C++20)	removes consecutive duplicate elements in a range (niebloid)
`unique_copy`	creates a copy of some range of elements that contains no consecutive duplicates (function template)
`ranges::unique_copy` (C++20)	creates a copy of some range of elements that contains no consecutive duplicates (niebloid)

Partitioning operations

Defined in header `<algorithm>`

`is_partitioned` (C++11)	determines if the range is partitioned by the given predicate (function template)
`ranges::is_partitioned` (C++20)	determines if the range is partitioned by the given predicate (niebloid)
`partition`	divides a range of elements into two groups (function template)
`ranges::partition` (C++20)	divides a range of elements into two groups (niebloid)
`partition_copy` (C++11)	copies a range dividing the elements into two groups (function template)
`ranges::partition_copy` (C++20)	copies a range dividing the elements into two groups (niebloid)
`stable_partition`	divides elements into two groups while preserving their relative order (function template)
`ranges::stable_partition` (C++20)	divides elements into two groups while preserving their relative order (niebloid)
`partition_point` (C++11)	locates the partition point of a partitioned range (function template)
`ranges::partition_point` (C++20)	locates the partition point of a partitioned range (niebloid)

Sorting operations

Defined in header `<algorithm>`

`is_sorted` (C++11)	checks whether a range is sorted into ascending order (function template)
`ranges::is_sorted` (C++20)	checks whether a range is sorted into ascending order (niebloid)
`is_sorted_until` (C++11)	finds the largest sorted subrange (function template)
`ranges::is_sorted_until` (C++20)	finds the largest sorted subrange (niebloid)
`sort`	sorts a range into ascending order (function template)
`ranges::sort` (C++20)	sorts a range into ascending order (niebloid)
`partial_sort`	sorts the first N elements of a range (function template)
`ranges::partial_sort` (C++20)	sorts the first N elements of a range (niebloid)
`partial_sort_copy`	copies and partially sorts a range of elements (function template)
`ranges::partial_sort_copy` (C++20)	copies and partially sorts a range of elements (niebloid)
`stable_sort`	sorts a range of elements while preserving order between equal elements (function template)
`ranges::stable_sort` (C++20)	sorts a range of elements while preserving order between equal elements (niebloid)
`nth_element`	partially sorts the given range making sure that it is partitioned by the given element (function template)

Binary search operations (on sorted ranges)

Defined in header <algorithm>

`lower_bound`	returns an iterator to the first element *not less* than the given value (function template)
`ranges::lower_bound` (C++20)	returns an iterator to the first element *not less* than the given value (niebloid)
`upper_bound`	returns an iterator to the first element *greater* than a certain value (function template)
`ranges::upper_bound` (C++20)	returns an iterator to the first element *greater* than a certain value (niebloid)
`binary_search`	determines if an element exists in a certain range (function template)
`ranges::binary_search` (C++20)	determines if an element exists in a certain range (niebloid)
`equal_range`	returns range of elements matching a specific key (function template)
`ranges::equal_range` (C++20)	returns range of elements matching a specific key (niebloid)

Other operations on sorted ranges

Defined in header <algorithm>

`merge`	merges two sorted ranges (function template)
`ranges::merge` (C++20)	merges two sorted ranges (niebloid)
`inplace_merge`	merges two ordered ranges in-place (function template)
`ranges::inplace_merge` (C++20)	merges two ordered ranges in-place (niebloid)

Set operations (on sorted ranges)

Defined in header <algorithm>

`includes`	returns true if one sequence is a subsequence of another (function template)
`ranges::includes` (C++20)	returns true if one sequence is a subsequence of another (niebloid)
`set_difference`	computes the difference between two sets (function template)
`ranges::set_difference` (C++20)	computes the difference between two sets (niebloid)
`set_intersection`	computes the intersection of two sets (function template)
`ranges::set_intersection` (C++20)	computes the intersection of two sets (niebloid)
`set_symmetric_difference`	computes the symmetric difference between two sets (function template)
`ranges::set_symmetric_difference` (C++20)	computes the symmetric difference between two sets (niebloid)
`set_union`	computes the union of two sets (function template)
`ranges::set_union` (C++20)	computes the union of two sets (niebloid)

Heap operations

Defined in header <algorithm>

`is_heap` (C++11)	checks if the given range is a max heap (function template)
`ranges::is_heap` (C++20)	checks if the given range is a max heap (niebloid)
`is_heap_until` (C++11)	finds the largest subrange that is a max heap (function template)
`ranges::is_heap_until` (C++20)	finds the largest subrange that is a max heap (niebloid)
`make_heap`	creates a max heap out of a range of elements (function template)
`ranges::make_heap` (C++20)	creates a max heap out of a range of elements (niebloid)
`push_heap`	adds an element to a max heap (function template)
`ranges::push_heap` (C++20)	adds an element to a max heap (niebloid)
`pop_heap`	removes the largest element from a max heap (function template)
`ranges::pop_heap` (C++20)	removes the largest element from a max heap (niebloid)
`sort_heap`	turns a max heap into a range of elements sorted in ascending order (function template)
`ranges::sort_heap` (C++20)	turns a max heap into a range of elements sorted in ascending order (niebloid)

Minimum/maximum operations

Defined in header <algorithm>

`max`	returns the greater of the given values (function template)
`ranges::max` (C++20)	returns the greater of the given values (niebloid)
`max_element`	returns the largest element in a range (function template)
`ranges::max_element` (C++20)	returns the largest element in a range (niebloid)
`min`	returns the smaller of the given values (function template)
`ranges::min` (C++20)	returns the smaller of the given values (niebloid)
`min_element`	returns the smallest element in a range (function template)
`ranges::min_element` (C++20)	returns the smallest element in a range (niebloid)
`minmax` (C++11)	returns the smaller and larger of two elements (function template)
`ranges::minmax` (C++20)	returns the smaller and larger of two elements (niebloid)
`minmax_element` (C++11)	returns the smallest and the largest elements in a range (function template)
`ranges::minmax_element` (C++20)	returns the smallest and the largest elements in a range (niebloid)
`clamp` (C++17)	clamps a value between a pair of boundary values (function template)
`ranges::clamp` (C++20)	clamps a value between a pair of boundary values (niebloid)

Comparison operations

Defined in header <algorithm>

`equal`	determines if two sets of elements are the same (function template)
`ranges::equal` (C++20)	determines if two sets of elements are the same (niebloid)
`lexicographical_compare`	returns true if one range is lexicographically less than another (function template)
`ranges::lexicographical_compare` (C++20)	returns true if one range is lexicographically less than another (niebloid)
`lexicographical_compare_three_way` (C++20)	compares two ranges using three-way comparison (function template)

Permutation operations

Defined in header <algorithm>

`is_permutation` (C++11)	determines if a sequence is a permutation of another sequence (function template)
`ranges::is_permutation` (C++20)	determines if a sequence is a permutation of another sequence (niebloid)
`next_permutation`	generates the next greater lexicographic permutation of a range of elements (function template)
`ranges::next_permutation` (C++20)	generates the next greater lexicographic permutation of a range of elements (niebloid)
`prev_permutation`	generates the next smaller lexicographic permutation of a range of elements (function template)
`ranges::prev_permutation` (C++20)	generates the next smaller lexicographic permutation of a range of elements (niebloid)

Numeric operations

Defined in header <numeric>

`iota` (C++11)	fills a range with successive increments of the starting value (function template)
`accumulate`	sums up a range of elements (function template)
`reduce` (C++17)	similar to `std::accumulate`, except out of order (function template)
`transform_reduce` (C++17)	applies an invocable, then reduces out of order (function template)
`inner_product`	computes the inner product of two ranges of elements (function template)
`adjacent_difference`	computes the differences between adjacent elements in a range (function template)
`partial_sum`	computes the partial sum of a range of elements (function template)
`inclusive_scan` (C++17)	similar to `std::partial_sum`, includes the ith input element in the ith sum (function template)
`exclusive_scan` (C++17)	similar to `std::partial_sum`, excludes the ith input element from the ith sum (function template)
`transform_inclusive_scan` (C++17)	applies an invocable, then calculates inclusive scan (function template)
`transform_exclusive_scan` (C++17)	applies an invocable, then calculates exclusive scan (function template)