FUNCTIONS IN STL

Non-modifying sequence operations:

for_each	Apply function to range
find	Find value in range
find_if	Find element in range
find_end	Find last subsequence in range
find_first_of	Find element from set in range
adjacent_find	Find equal adjacent elements in range
count	Count appearances of value in range
count_if	Return number of elements in range satisfying condition
mismatch	Return first position where two ranges differ
equal	Test whether the elements in two ranges are equal
search	Find subsequence in range
search_n	Find succession of equal values in range

Modifying sequence operations:

copy	Copy range of elements
copy_backward	Copy range of elements backwards
swap	Exchange values of two objects
swap_ranges	Exchange values of two ranges
iter_swap	Exchange values of objects pointed by two iterators
transform	Apply function to range
replace	Replace value in range
replace_if	Replace values in range
replace_copy	Copy range replacing value
replace_copy_if	Copy range replacing value
fill	Fill range with value
fill_n	Fill sequence with value
generate	Generate values for range with function
generate_n	Generate values for sequence with function
remove	Remove value from range
remove_if	Remove elements from range
remove_copy	Copy range removing value

remove_copy_if	Copy range removing values
unique	Remove consecutive duplicates in range
unique_copy	Copy range removing duplicates
reverse	Reverse range
reverse_copy	Copy range reversed
rotate	Rotate elements in range
rotate_copy	Copy rotated range
random_shuffle	Rearrange elements in range randomly
partition	Partition range in two
stable_partition	Partition range in two - stable ordering

Sorting:

sort	Sort elements in range
stable_sort	Sort elements preserving order of equivalents
partial_sort	Partially Sort elements in range
partial_sort_copy	Copy and partially sort range
nth_element	Sort element in range

Binary search:

lower_bound	Return iterator to lower bound
upper_bound	Return iterator to upper bound
equal_range	Get subrange of equal elements
binary_search	Test if value exists in sorted array

Merge:

merge	Merge sorted ranges
inplace_merge	Merge consecutive sorted ranges
includes	Test whether sorted range includes another sorted range
set_union	Union of two sorted ranges
set_intersection	Intersection of two sorted ranges
set_difference	Difference of two sorted ranges
set_symmetric_difference	Symmetric difference of two sorted ranges

Heaps:

push_heap	Push element into heap range
pop_heap	Pop element from heap range
make_heap	Make heap from range
sort_heap	Sort elements of heap

Min and Max:

min	Return the lesser of two arguments
max	Return the greater of two arguments
min_element	Return smallest element in range
max_element	Return largest element in range

lexicographical_compare	Lexicographical less-than comparison

Permutations:

next_permutation	Transform range to next permutation
prev_permutation	Transform range to previous permutation

Function Prototypes

Non-modifying sequence operations:

for_each	Apply function to range
	template <class II, class UF> UF for_each (II first, II last, UF f) – apply the unary function or the function object f to each element in the range [first, last) and return f
find	Find value in range
	template <class II, class T> II find (II first, II last, const T& x) – return an input iterator pointing to the first element in the range [first, last) that matches the value x, or return last if the value is not found
find_if	Find element in range
	template <class II, class UP> II find_if (II first, II last, UP p) – return an input iterator pointing to the first element in the range [first, last), for which the unary predicate or the function object p is true, or return last if p is false for all the elements in the range
find_end	Find last subsequence in range
	template <class FI1, class FI2, class BP) FI1 find_end (FI1 first1, FI1 last1, FI2 first2, FI2 last2, BP p =equal_to <typename FI1::value_type> ()) – search the sequence of elements in the second range [first2, last2) as a subsequence of the elements in the first range [first1, last1), for which the binary predicate or the function object p is true for the corresponding elements in those ranges, and if a match is found, return a forward iterator pointing to the first element of the last match in the first range
find_first_of	Find element from set in range
	template <class FI1, class FI2, class BP> FI1 find_first_of (FI1 first1, FI1 last1, FI2 first2, FI2 last2, BP p =equal_to <typename FI1::value_type> ()) – search for the first element in the first range [first1, last1) against the elements in the second range [first2, last2), for which the binary predicate or the function object p is true for the corresponding elements in those ranges, and if a match is found, return a forward iterator pointing to the first element in the first range
adjacent_find	Find equal adjacent elements in range
	template <class FI, class BP> FI adjacent_find (FI first, FI last, BP p =equal_to <typename FI::value_type> ()) – search the first occurrence of two consecutive elements in the range [first, last), for which the binary predicate or the function object p is true, and if a match is found, return a forward iterator pointing to the first of these two elements
count	Count appearances of value in range
	template <class II, class T> ptrdiff_t count (II first, II last, const T& x) – return the total number of occurrences of the elements with the value x in the range [first, last)
count_if	Return number of elements in range satisfying condition
	template <class II, class P> ptrdiff_t count_if (II first, II last, UP p) – return the total

	number of occurrences of the elements in the range [first, last) for which the unary predicate or the function object p is true
mismatch	Return first position where two ranges differ
	template <class II1, class II2, class BP> pair <II1, II2> mismatch (II1 first1, II1 last1, II2 first2, BP p =equal_to <typename II1::value_type> ()) – compare the elements in the range [first1, last1) against those in the range beginning at first2, and return a pair of input iterators pointing to the first elements in each range for which the binary predicate or the function object p is false
equal	Test whether the elements in two ranges are equal
	template <class II1, class II2, class BP> bool equal (II1 first1, II1 last1, II2 first2, BP p =equal_to <typename II1::value_type> ()) – compare the elements in the range [first1, last1) with those in the range beginning at first2 and return true if the binary predicate or the function object p is true for all the corresponding elements in those ranges
search	Find subsequence in range
	template <class FI1, class FI2, class BP> FI1 search (FI1 first1, FI1 last1, FI2 first2, FI2 last2, BP p =equal_to <typename FI1::value_type> ()) – search for the sequence of the elements in the second range [first2, last2) as a subsequence of the elements in the first range [first1, last1), for which the binary predicate or the function object p is true for all the corresponding elements in those ranges, and if a match is found, return a forward iterator pointing to the first element in the first range
search_n	Find succession of equal values in range
	template <class FI, class N, class T, class BP> FI search_n (FI first, FI last, N n, const T& x, BP p =equal_to <typename FI::value_type> ()) – search the range [first, last) for the succession of n elements, for which the binary predicate or the function object p is true when each of those elements is compared with x, and if a match is found, return a forward iterator pointing to the first of these elements

Modifying sequence operations:

copy	Copy range of elements
	template <class II, class OI> OI copy (II first1, II last1, OI first2) – copy the elements in the range [first1, last1) into the range beginning at first2, and return an output iterator that follows the last element in the destination range
copy_backward	Copy range of elements backwards
	template <class BI1, class BI2> BI2 copy_backward (BI1 first1, BI1 last1, BI2 last2) – copy the elements in the range [first1, last1), starting from the last element in the range and then following backwards by the elements until first1 is reached, into the range whose last element is just before the element referred by the bidirectional iterator last2, and return a bidirectional iterator to the first element in the destination range
swap	Exchange values of two objects
	template <class T> void swap (T& x, T& y) – swap the values of the objects x and y

swap_ranges	Exchange values of two ranges
	template <class FI1, class FI2> FI2 swap_ranges (FI1 first1, FI1 last1, FI2 first2) – swap the values of each element in the range [first1, last1) with those in the range starting at first2, and return a forward iterator that follows the last element swapped in the second range
iter_swap	Exchange values of objects pointed by two iterators
	template <class FI1, class FI2> void iter_swap (FI1 i, FI2 j) – swap the values of the objects pointed by the forward iterators i and j
transform	Apply function to range
	template <class II, class OI, class UF> OI transform (II first1, II last1, OI first2, UF f) – apply the unary function or the function object f to all elements in the range [first1, last1) and store each returned value in the range starting at first2, and return an output iterator pointing to the element follows the last element in the destination range
	template <class II1, class II2, class OI, class BF> OI transform (II1 first1, II1 last1, II2 first2, OI first3, BF f) – apply the binary function or the function object f to all pairs of elements with the first element of the pair is from the range [first1, last1) and its second element is from the range starting at first2, and store each returned value into the range starting at first3, and return an output iterator pointing to the last element in the destination range
replace	Replace value in range
	template <class FI, class T> void replace (FI first, FI last, const T& old, const T& new) – set all the elements in the range [first, last) with the value old to the value new
replace_if	Replace values in range
	template <class FI, class UP, class T> void replace_if (FI first, FI last, UP p, const T& new) – set all the elements in the range [first, last), for which the unary predicate or the function object p is true, to the value new
replace_copy	Copy range replacing value
	template <class II, class OI, class T> OI replace_copy (II first1, II last1, OI first2, const T& old, const T& new) – copy the values in the range [first1, last1) to the positions in the range starting at first2, replacing all instances of the value old with the value new, and return an output iterator that follows the last element in the destination range
replace_copy_if	Copy range replacing value
	template <class II, class OI, class UP, class T> OI replace_copy_if (II first1, II last1, OI first2, UP p, const T& new) – copy the values in the range [first1, last1) to the positions in the range starting at first2, replacing all instances of the values, for which the unary predicate or the function object p is true, to the value new, and return an output iterator that follows the last element in the destination range
fill	Fill range with value
	template <class FI, class T> void fill (FI first, FI last, const T& x) – set the values of the elements in the range [first, last) to the value x

fill_n	Fill sequence with value
	template <class OI, class N, class T> void fill_n (OI first, N n, const T& x) – set the values of the first n elements in the range starting at first to the value x
generate	Generate values for range with function
	template <class OI, class G> void generate (OI first, OI last, G g) – set the values of the elements in the range [first, last) to the values returned by the successive calls of the generator function or the function object g
generate_n	Generate values for sequence with function
	template <class OI, class N, class G> void generate_n (OI first, N n, G g) – set the values of the first n elements in the range starting at first to the values returned by the successive calls of the generator function or the function object g
remove	Remove value from range
	template <class FI, class T> FI remove (FI first, FI last, const T& x) – remove the elements with the value x from the range [first, last) and return a forward iterator to the new end of the range
remove_if	Remove elements from range
	template <class FI, class UP> FI remove_if (FI first, FI last, UP p) – remove the elements in the range [first, last), for which the unary predicate or the function object p is true and return a forward iterator to the new end of the range
remove_copy	Copy range removing value
	template <class II, class OI, class T> OI remove_copy (II first1, II last1, OI first2, const T& x) – copy the elements in the range [first1, last1) to the positions in the range starting at first2, except the elements whose values equal to x, and return an output iterator to the end of the destination range
remove_copy_if	Copy range removing values
	template <class II, class OI, class UP> OI remove_copy_if (II first1, II last1, OI first2, UP p) – copy the elements in the range [first1, last1) to the positions in the range starting at first2, except the elements for which the unary predicate or the function object p is true, and return an output iterator to the end of the destination range
unique	Remove consecutive duplicates in range
	template <class FI, class BP> FI unique (FI first, FI last, BP p =equal_to <typename FI::value_type> ()) – move all the elements in the range [first, last) to the front of the range, except for the adjacent consecutive elements for which the binary predicate p is true, and return a forward iterator to the new end of the range
unique_copy	Copy range removing duplicates
	template <class II, class OI, class BP> OI unique_copy (II first1, II last1, OI first2, BP p =equal_to <typename II::value_type> ()) – copy all the elements in the range [first1, last1) to the positions in the range starting at first2, except for the adjacent consecutive elements for which the binary predicate or the function object p is true, and return an output iterator to the end of the destination range

reverse	Reverse range
	template <class BI> void reverse (BI first, BI last) – reverse the order of elements in the range [first, last)
reverse_copy	Copy range reversed
	template <class BI, class I2> OI reverse_copy (BI first1, BI last1, OI first2) – copy the elements in the range [first1, last1) to the positions in the range starting at first2, but reversing the order of the elements, and return an output iterator to the end of the destination range
rotate	Rotate elements in range
	template <class FI> void rotate (FI first, FI middle, FI last) – rotate the order of the elements in the range [first, last) in such a way that the element referred by the forward iterator middle becomes the new first element
rotate_copy	Copy rotated range
	template <class FI, class OI> OI rotate_copy (FI first1, FI middle1, FI last1, OI first2) – copy the elements in the range [first1, last1) to the positions in the range starting at first2, but rotating the order of the elements in such a way that the element referred by the forward iterator middle1 becomes the first element in the destination range, and return an output iterator to the end of the destination range
random_shuffle	Rearrange elements in range randomly
	template <class RI, class G> void random_shuffle (RI first, RI last, G& g =Rand) – swap the value of each element in the range [first, last) with some other randomly chosen element by the random number generator g, where Rand is defined as ptrdiff_t Rand (ptrdiff_t i) { return rand () % i; }
partition	Partition range in two
	template <class BI, class UP> BI partition (BI first, BI last, UP p) – rearrange the elements in the range [first, last), in such a way that all the elements for which the unary predicate or the function object p is true precede all those for which p is false, and return a bidirectional iterator referring to the first element in the second group
stable_partition	Partition range in two - stable ordering
	template <class BI, class UP> BI stable_partition (BI first, BI last, UP p) – same as the function partition, but it keeps the relative order of the elements in each group

Sorting:

sort	Sort elements in range
	template <class RI, class C> void sort (RI first, RI last, C cmp =less <typename RI::value_type> ()) – sort the elements in the range [first, last) for which the order is determined by the compare function or the function object cmp
stable_sort	Sort elements preserving order of equivalents
	template <class RI, class C> void stable_sort (RI first, RI last, C cmp =less <typename

	RI::value_type> ()) – same as the function sort, but it keeps the relative order of the elements
partial_sort	Partially sort elements in range
	template <class RI, class C> void partial_sort (RI first, RI middle, RI last, C cmp =less <typename RI::value_type> ()) – rearrange the elements in the range [first, last), in such a way that the subrange [first, middle) contains the elements in the order determined by the compare function or the function object cmp, and the subrange [middle, last) contains the rest of the elements without any specific order
partial_sort_copy	Copy and partially sort range
	template <class II, class RI, class C> RI partial_sort_copy (II first1, II last1, RI first2, RI last2, C cmp =less <typename II::value_type> ()) – copy n elements, for which the order is determined by the compare function or the function object cmp, from the range [first1, last1) to the range [first2, first2+n), where n is the smaller of (last1–first1) and (last2–first2), sorting the copied elements in the order determined by cmp, and return a random-access iterator to (first2 + n)
nth_element	Sort element in range
	template <class RI, class C> void nth_element (RI first, RI nth, I last, C cmp =less <typename RI::value_type> ()) – rearrange the elements in the range [first, last), in such a way that the element at the position referred by the random-access iterator nth is the element that would be in that position in a sorted sequence determined by the compare function or the function object cmp with the elements preceding it come before and the elements following it come after it in the order determined by cmp, but neither the elements preceding it nor the elements following it are granted to be ordered

Binary search:

lower_bound	Return iterator to lower bound
	template <class FI, class T, class C> FI lower_bound (FI first, FI last, const T& x, C cmp =less <typename FI::value_type> ()) – return a forward iterator pointing to the position of the first element in the sorted range [first, last), whose value is greater than or equal to x in the order determined by the compare function or the function object cmp. If no such element exists, it returns the forward iterator last.
upper_bound	Return iterator to upper bound
	template <class FI, class T, class C> FI upper_bound (FI first, FI last, const T& x, C cmp =less <typename FI::value_type> ()) – return a forward iterator pointing to the position of the first element in the sorted range [first, last), whose value is greater than x in the order determined by the compare function or the function object cmp. If no such element exists, it returns the forward iterator last.
equal_range	Get subrange of equal elements
	template <class FI, class T, class C> pair <FI, FI> equal_range (FI first, FI last, const T& x, C cmp =less <typename FI::value_type> ()) – return a pair of forward iterators pointing to the bounds of the largest subrange in the sorted range [first, last) that

	includes all the elements with values equal to x, where the equality is determined by the compare function or the function object cmp
binary_search	Test if value exists in sorted array
	template <class FI, class T, class C> bool binary_search (FI first, FI last, const T& x, C cmp =less <typename FI::value_type> ()) – return true if the value of an element in the sorted range [first, last) equals to x, where the equality is determined by the compare function or the function object cmp

Merge:

merge	Merge sorted ranges
	template <class II1, class II2, class OI, class C> OI merge (II1 first1, II1 last1, II2 first2, II2 last2, OI first3, C cmp =less <typename II1::value_type> ()) – combine the elements in the sorted ranges [first1, last1) and [first2, last2) into a new range starting at first3 with its elements sorted in the order determined by the compare function or the function object cmp, and return an output iterator pointing past to the last element in the resulting sequence
inplace_merge	Merge consecutive sorted ranges
	template <class BI, class C> void inplace_merge (BI first, BI middle, BI last, C cmp =less <typename BI::value_type> ()) – merge the elements in two consecutive sorted ranges in [first, middle) and [middle, last), putting the result into the range [first, last) with its elements sorted in the order determined by the compare function or the function object cmp
includes	Test whether sorted range includes another sorted range
	template <class II1, class II2, class C> bool includes (II1 first1, II1 last1, II2 first2, II2 last2, C cmp =less <typename II1::value_type> ()) – return true if all the elements in the sorted range [first2, last2) equal to some of the elements in the sorted range [first1, last1), where the equality is determined by the compare function or the function object cmp
set_union	Union of two sorted ranges
	template <class II1, class II2, class OI, class C> OI set_union (II1 first1, II1 last1, II2 first2, II2 last2, OI first3, C cmp =less <typename II1::value_type> ()) – construct a range of elements in the order determined by the compare function or the function object cmp, starting at the location pointed by the output iterator first3 with the *set union* of the sorted elements in the ranges [first1, last1) and [first2, last2), and return an output iterator to the end of the constructed range
set_intersection	Intersection of two sorted ranges
	template <class II1, class II2, class OI, class C> OI set_intersection (II1 first1, II1 last1, II2 first2, II2 last2, OI first3, C cmp =less <typename II1::value_type> ()) – construct a range of elements in the order determined by the compare function or the function object cmp, starting at the location pointed by the output iterator first3 with the *set intersection* of the sorted elements in the ranges [first1, last1) and [first2, last2), and return an output iterator to the end of the constructed range

set_difference	Difference of two sorted ranges
	template <class II, class II, class OI, class C> OI set_difference (II1 first1, II1 last1, II2 first2, II2 last2, OI first3, C cmp =less <typename II1::value_type> ()) – construct a range of elements in the order determined by the compare function or the function object cmp, starting at the location pointed by the output iterator first3 with the *set difference* of the sorted elements in the ranges [first1, last1) and [first2, last2), and return an output iterator to the end of the constructed range
set_symmetric_difference	Symmetric difference of two sorted ranges
	template <class II1, class II2, class OI, class C> OI set_symmetric_difference (II1 first1, II1 last1, II2 first2, II2 last2, OI first3, C cmp =less <typename II1::value_type> ()) – construct a range of elements in the order determined by the compare function or the function object cmp, starting at the location pointed by the output iterator first3 with the *set symmetric difference* of the sorted elements in the ranges [first1, last1) and [first2, last2), and return an output iterator to the end of the constructed range

Heaps:

push_heap	Push element into heap range
	template <class RI, class C> void push_heap (RI first, RI last, C cmp =less <typename RI::value_type> ()) – given a heap range [first, last–1), extend the range to [first, last) by placing the value at the location referred by the random-access iterator (last–1) into the heap, where the order of the elements is determined by the compare function or the function object cmp
pop_heap	Pop element from heap range
	template <class RI, class C> void pop_heap (RI first, RI last, C cmp =less <typename RI::value_type> ()) – rearrange the elements in the range [first, last) in such a way that the part is considered a heap is shortened by one removing the largest element determined by the compare function or the function object cmp
make_heap	Make heap from range
	template <class RI, class C> void make_heap (RI first, RI last, C cmp =less <typename RI::value_type> ()) – rearrange the elements in the range [first, last) in such a way that they form a heap, where the order of the elements is determined by the compare function or the function object cmp
sort_heap	Sort elements of heap
	template <class RI, class C> void sort_heap (RI first, RI last, C cmp =less <typename RI::value_type> ()) – rearrange the elements in the heap range [first, last) in such a way that they form a sorted range, where the order of the sort is determined by the compare function or the function object cmp

Min and Max:

min	Return the lesser of two arguments

	template <class T, class C> const T& min (const T& x, const T& y, C cmp =less <typename I::value_type> ()) – return the lesser of the values x and y, where the lesser value is determined by the compare function or the function object cmp
max	Return the greater of two arguments
	template <class T, class C> const T& max (const T& x, const T& y, C cmp =less <typename I::value_type> ()) – return the greater of the values x and y, where the greater value is determined by the compare function or the function object cmp
min_element	Return smallest element in range
	template <class FI, class C> FI min_element (FI first, FI last, C cmp =less <typename FI::value_type> ()) – return a forward iterator pointing to the element with the smallest value in the range [first, last), where the smallest value is determined by the compare function or the function object cmp
max_element	Return largest element in range
	template <class FI, class C> FI max_element (FI first, FI last, C cmp =less <typename FI::value_type> ()) – return a forward iterator pointing to the element with the largest value in the range [first, last), where the largest value is determined by the compare function or the function object cmp

lexicographical_compare	Lexicographical less-than comparison
	template <class II1, class II2, class C> bool lexicographical_compare (II1 first1, II1 last1, II2 first2, II2 last2, C cmp =less <typename II1::value_type> ()) – return true if the elements in the range [first1, last1) is lexicographically less than the elements in the range [first2, last2), where the order of elements is determined by the compare function or the function object cmp

Permutations:

next_permutation	Transform range to next permutation
	template <class BI, class C> bool next_permutation (BI first, BI last, C cmp =less <typename BI::value_type> ()) – rearrange the elements in the range [first, last) lexicographically to the next permutation of the elements and return true; if the current permutation is already the last one, rearrange the elements according to the first permutation and return false, where the order of the elements is determined by the compare function or the function object cmp
prev_permutation	Transform range to previous permutation
	template <class BI, class C> bool prev_permutation (BI first, BI last, C cmp =less <typename BI::value_type> ()) – rearrange the elements in the range [first, last) lexicographically to the previous permutation of the elements and return true; if the current permutation is already the first one, rearrange the elements according to the last permutation and return false, where the order of the elements is determined by the compare function or the function object cmp