<algorithm>

1. all_of Test condition on all elements in range
template <class InputIterator, class UnaryPredicate> bool all_of (InputIterator first, InputIterator last, UnaryPredicate pred);

2. any_of Test if any element in range fulfills condition
template <class InputIterator, class UnaryPredicate> bool any_of (InputIterator first, InputIterator last, UnaryPredicate pred);

3. none_of Test if no elements fulfill condition
template <class InputIterator, class UnaryPredicate> bool none_of (InputIterator first, InputIterator last, UnaryPredicate pred);

4. for_each Apply function to range
template <class InputIterator, class Function> Function for_each (InputIterator first, InputIterator last, Function fn);

5. find Find value in range
template <class InputIterator, class T> InputIterator find (InputIterator first, InputIterator last, const T& val);

6. find_if Find element in range
template <class InputIterator, class UnaryPredicate> InputIterator find_if (InputIterator first, InputIterator last, UnaryPredicate pred);

7. find_if_not Find element in range (negative condition)
template <class InputIterator, class UnaryPredicate> InputIterator find_if_not (InputIterator first, InputIterator last, UnaryPredicate pred);

8. find_end Find last subsequence in range
template <class ForwardIterator1, class ForwardIterator2> ForwardIterator1 find_end (ForwardIterator1 first1, ForwardIterator1 last1, ForwardIterator2 first2, ForwardIterator2 last2);

9. find_first_of Find element from set in range
template <class InputIterator, class ForwardIterator> ForwardIterator1 find_first_of(InputIterator first1, InputIterator last1, ForwardIterator first2, ForwardIterator last2);
template <class InputIterator, class ForwardIterator, class BinaryPredicate> ForwardIterator1 find_first_of(InputIterator first1, InputIterator last1, ForwardIterator first2, ForwardIterator last2, BinaryPredicate pred);
template <class InputIterator, class ForwardIterator> ForwardIterator1 find_first_of(InputIterator first1, InputIterator last1, ForwardIterator first2, ForwardIterator last2);

10. adjacent_find Find equal adjacent elements in range
template <class ForwardIterator> ForwardIterator adjacent_find (ForwardIterator first, ForwardIterator last);

11. count Count appearances of value in range
template <class InputIterator, class T> typename iterator_traits<InputIterator>::difference_type count (InputIterator first, InputIterator last, const T& val);

12. count_if Return number of elements in range satisfying condition
template <class InputIterator, class Predicate> typename iterator_traits<InputIterator>::difference_type count_if (InputIterator first, InputIterator last, UnaryPredicate pred);

13. mismatch Return first position where two ranges differ
template <class InputIterator1, class InputIterator2> pair<InputIterator1, InputIterator2> mismatch (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2);

14. equal Test whether the elements in two ranges are equal
template <class InputIterator1, class InputIterator2> bool equal (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2);

15. is_permutation Test whether range is permutation of another
template <class ForwardIterator1, class ForwardIterator2> bool is_permutation (ForwardIterator1 first1, ForwardIterator1 last1, ForwardIterator2 first2);

16. search Search range for subsequence
template <class ForwardIterator1, class ForwardIterator2> ForwardIterator1 search (ForwardIterator1 first1, ForwardIterator1 last1, ForwardIterator2 first2, ForwardIterator2 last2);

17. search_n Search range for elements
template <class ForwardIterator, class Size, class T> ForwardIterator search_n (ForwardIterator first, ForwardIterator last, Size count, const T& val);

18. copy Copy range of elements
template <class InputIterator, class OutputIterator> OutputIterator copy (InputIterator first, InputIterator last, OutputIterator result);

19. copy_n Copy elements
template <class InputIterator, class Size, class OutputIterator> OutputIterator copy_n (InputIterator first, Size n, OutputIterator result);

20. copy_if Copy certain elements of range
template <class InputIterator, class OutputIterator, class UnaryPredicate> OutputIterator copy_if (InputIterator first, InputIterator last, OutputIterator result, UnaryPredicate pred);

21. copy_backward Copy range of elements backward
template <class BidirectionalIterator1, class BidirectionalIterator2> BidirectionalIterator2 copy_backward (BidirectionalIterator1 first, BidirectionalIterator1 last, BidirectionalIterator2 result);

22. move Move range of elements
template <class InputIterator, class OutputIterator> OutputIterator move (InputIterator first, InputIterator last, OutputIterator result);

23. move_backward Move range of elements backward
template <class BidirectionalIterator1, class BidirectionalIterator2> BidirectionalIterator2 move_backward (BidirectionalIterator1 first, BidirectionalIterator1 last, BidirectionalIterator2 result);

24. swap Exchange values of two objects
template <class T> void swap(T& a, T& b)
noexcept(is_nothrow_move_constructible<T>::value &&
is_nothrow_move_assignable<T>::value);
template <class T, size_t N> void swap(T(&a)[N], T(&b)[N])
noexcept(noexcept(swap(*a,*b)));

25. swap_ranges Exchange values of two ranges
template <class ForwardIterator1, class ForwardIterator2> ForwardIterator2 swap_ranges (ForwardIterator1 first1, ForwardIterator1 last1, ForwardIterator2 first2);

26. iter_swap Exchange values of objects pointed by two iterators
template <class ForwardIterator1, class ForwardIterator2> void iter_swap (ForwardIterator1 a, ForwardIterator2 b);

27. transform Transform range
template <class InputIterator, class OutputIterator, class UnaryOperation> OutputIterator transform (InputIterator first1, InputIterator last1, OutputIterator result, UnaryOperation op);

28. replace Replace value in range
template <class ForwardIterator, class T> void replace (ForwardIterator first, ForwardIterator last, const T& old_value, const T& new_value);

29. replace_if Replace values in range
template <class ForwardIterator, class UnaryPredicate, class T> void replace_if (ForwardIterator first, ForwardIterator last, UnaryPredicate pred, const T& new_value);

30. replace_copy Copy range replacing value
template <class InputIterator, class OutputIterator, class T> OutputIterator replace_copy (InputIterator first, InputIterator last, OutputIterator result, const T& old_value, const T& new_value);

31. replace_copy_if Copy range replacing value
template <class InputIterator, class OutputIterator, class UnaryPredicate, class T> OutputIterator replace_copy_if (InputIterator first, InputIterator last, OutputIterator result, UnaryPredicate pred, const T& new_value);

32. fill Fill range with value
template <class ForwardIterator, class T> void fill (ForwardIterator first, ForwardIterator last, const T& val);

33. fill_n Fill sequence with value
template <class OutputIterator, class Size, class T> OutputIterator fill_n(OutputIterator first, Size n, const T& val);

34. generate Generate values for range with function
template <class ForwardIterator, class Generator> void generate (ForwardIterator first, ForwardIterator last, Generator gen);

35. generate_n Generate values for sequence with function
template <class OutputIterator, class Size, class Generator> OutputIterator generate_n(OutputIterator first, Size n, Generator gen);

36. remove Remove value from range
template <class ForwardIterator, class T> ForwardIterator remove(ForwardIterator first, ForwardIterator last, const T& val)
{ ForwardIterator result = first;

37. remove_if Remove elements from range
template <class ForwardIterator, class UnaryPredicate> ForwardIterator remove_if(ForwardIterator first, ForwardIterator last, UnaryPredicate pred)

38. remove_copy Copy range removing value
template <class InputIterator, class OutputIterator, class T> OutputIterator remove_copy (InputIterator first, InputIterator last, OutputIterator result, const T& val);

39. remove_copy_if Copy range removing values
template <class InputIterator, class OutputIterator, class UnaryPredicate> OutputIterator remove_copy_if (InputIterator first, InputIterator last, OutputIterator result, UnaryPredicate pred);

40. unique Remove consecutive duplicates in range
template <class ForwardIterator> ForwardIterator unique (ForwardIterator first, ForwardIterator last);

41. unique_copy Copy range removing duplicates
template <class InputIterator, class OutputIterator> OutputIterator unique_copy (InputIterator first, InputIterator last, OutputIterator result);

42. reverse Reverse range
template <class BidirectionalIterator> void reverse (BidirectionalIterator first, BidirectionalIterator last);

43. reverse_copy Copy range reversed
template <class BidirectionalIterator, class OutputIterator> OutputIterator reverse_copy (BidirectionalIterator first, BidirectionalIterator last, OutputIterator result);

44. rotate Rotate left the elements in range
template <class ForwardIterator> ForwardIterator rotate(ForwardIterator first, ForwardIterator middle, ForwardIterator last);

45. rotate_copy Copy range rotated left
template <class ForwardIterator, class OutputIterator> OutputIterator rotate_copy (ForwardIterator first, ForwardIterator middle, ForwardIterator last, OutputIterator result);

46. random_shuffle Randomly rearrange elements in range
template <class RandomAccessIterator> void random_shuffle(RandomAccessIterator first, RandomAccessIterator last);
template <class RandomAccessIterator, class RandomNumberGenerator> void random_shuffle(RandomAccessIterator first, RandomAccessIterator last, RandomNumberGenerator&& gen);
template <class RandomAccessIterator> void random_shuffle(RandomAccessIterator first, RandomAccessIterator last);

47. shuffle Randomly rearrange elements in range using generator
template <class RandomAccessIterator, class URNG> void shuffle (RandomAccessIterator first, RandomAccessIterator last, URNG&& g);

48. is_partitioned Test whether range is partitioned
template <class InputIterator, class UnaryPredicate> bool is_partitioned (InputIterator first, InputIterator last, UnaryPredicate pred);

49. partition Partition range in two
template <class ForwardIterator, class UnaryPredicate> ForwardIterator partition(ForwardIterator first, ForwardIterator last, UnaryPredicate pred);

50. stable_partition Partition range in two - stable ordering
template <class BidirectionalIterator, class UnaryPredicate> BidirectionalIterator stable_partition (BidirectionalIterator first, BidirectionalIterator last, UnaryPredicate pred);

51. partition_copy Partition range into two
template <class InputIterator, class OutputIterator1, class OutputIterator2, class UnaryPredicate pred> pair<OutputIterator1,OutputIterator2> partition_copy (InputIterator first, InputIterator last, OutputIterator1 result_true, OutputIterator2 result_false, UnaryPredicate pred);

52. partition_point Get partition point
template <class ForwardIterator, class UnaryPredicate> ForwardIterator partition_point (ForwardIterator first, ForwardIterator last, UnaryPredicate pred);

53. sort Sort elements in range
template <class RandomAccessIterator> void sort (RandomAccessIterator first, RandomAccessIterator last);

54. stable_sort Sort elements preserving order of equivalents
template <class RandomAccessIterator> void stable_sort (RandomAccessIterator first, RandomAccessIterator last); template <class RandomAccessIterator, class Compare> void stable_sort (RandomAccessIterator first, RandomAccessIterator last, Compare comp);

55. partial_sort Partially sort elements in range
template <class RandomAccessIterator> void partial_sort (RandomAccessIterator first, RandomAccessIterator middle, RandomAccessIterator last);

56. partial_sort_copy Copy and partially sort range
template <class InputIterator, class RandomAccessIterator> RandomAccessIterator partial_sort_copy (InputIterator first,InputIterator last, RandomAccessIterator result_first, RandomAccessIterator result_last);

57. is_sorted Check whether range is sorted
template <class ForwardIterator> bool is_sorted (ForwardIterator first, ForwardIterator last);

58. is_sorted_until Find first unsorted element in range
template <class ForwardIterator> ForwardIterator is_sorted_until (ForwardIterator first, ForwardIterator last);

59. nth_element Sort element in range
template <class RandomAccessIterator> void nth_element (RandomAccessIterator first, RandomAccessIterator nth, RandomAccessIterator last);

60. lower_bound Return iterator to lower bound
template <class ForwardIterator, class T> ForwardIterator lower_bound (ForwardIterator first, ForwardIterator last, const T& val);

61. upper_bound Return iterator to upper bound
template <class ForwardIterator, class T> ForwardIterator upper_bound (ForwardIterator first, ForwardIterator last, const T& val);

62. equal_range Get subrange of equal elements
template <class ForwardIterator, class T> pair<ForwardIterator,ForwardIterator> equal_range (ForwardIterator first, ForwardIterator last, const T& val);

63. binary_search Test if value exists in sorted sequence
template <class ForwardIterator, class T> bool binary_search (ForwardIterator first, ForwardIterator last, const T& val);

64. merge Merge sorted ranges
template <class InputIterator1, class InputIterator2, class OutputIterator> OutputIterator merge (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, InputIterator2 last2, OutputIterator result);

65. inplace_merge Merge consecutive sorted ranges
template <class BidirectionalIterator> void inplace_merge (BidirectionalIterator first, BidirectionalIterator middle, BidirectionalIterator last);

66. includes Test whether sorted range includes another sorted range
template <class InputIterator1, class InputIterator2> bool includes (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, InputIterator2 last2); template <class InputIterator1, class InputIterator2, class Compare> bool includes (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, InputIterator2 last2, Compare comp);

67. set_union Union of two sorted ranges
template <class InputIterator1, class InputIterator2, class OutputIterator> OutputIterator set_union (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, InputIterator2 last2, OutputIterator result);

68. set_intersection Intersection of two sorted ranges
template <class InputIterator1, class InputIterator2, class OutputIterator> OutputIterator set_intersection (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, InputIterator2 last2, OutputIterator result);

69. set_difference Difference of two sorted ranges
template <class InputIterator1, class InputIterator2, class OutputIterator> OutputIterator set_difference (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, InputIterator2 last2, OutputIterator result);

70. set_symmetric_difference Symmetric difference of two sorted ranges
template <class InputIterator1, class InputIterator2, class OutputIterator> OutputIterator set_symmetric_difference (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, InputIterator2 last2, OutputIterator result);

71. push_heap Push element into heap range
template <class RandomAccessIterator> void push_heap (RandomAccessIterator first, RandomAccessIterator last);

72. pop_heap Pop element from heap range
template <class RandomAccessIterator> void pop_heap (RandomAccessIterator first, RandomAccessIterator last);

73. make_heap Make heap from range
template <class RandomAccessIterator> void make_heap (RandomAccessIterator first, RandomAccessIterator last);

74. sort_heap Sort elements of heap
template <class RandomAccessIterator> void sort_heap (RandomAccessIterator first, RandomAccessIterator last);

75. is_heap Test if range is heap
template <class RandomAccessIterator> bool is_heap (RandomAccessIterator first, RandomAccessIterator last);

76. is_heap_until Find first element not in heap order
template <class RandomAccessIterator> RandomAccessIterator is_heap_until (RandomAccessIterator first, RandomAccessIterator last);

77. min Return the smallest
template <class T> const T& min(const T& a, const T& b);
template <class T, class Compare> const T& min(const T& a, const T& b, Compare comp);
template <class T> T min(initializer_list<T> il);
template <class T, class Compare> T min(initializer_list<T> il, Compare comp);
template <class T> const T& min(const T& a, const T& b);

78. max Return the largest
template <class T> const T& max(const T& a, const T& b);
template <class T, class Compare> const T& max(const T& a, const T& b, Compare comp);
template <class T> T max(initializer_list<T> il);
template <class T, class Compare> T max(initializer_list<T> il, Compare comp);
template <class T> const T& max(const T& a, const T& b);

79. minmax Return smallest and largest elements
template <class T> pair <const T&,const T&> minmax(const T& a, const T& b);
template <class T, class Compare> pair <const T&,const T&> minmax(const T& a, const T& b, Compare comp);
template <class T> pair<T,T> minmax(initializer_list<T> il);
template <class T, class Compare> pair<T,T> minmax(initializer_list<T> il, Compare comp);
template <class T> pair <const T&,const T&> minmax(const T& a, const T& b);

80. min_element Return smallest element in range
template <class ForwardIterator> ForwardIterator min_element (ForwardIterator first, ForwardIterator last);

81. max_element Return largest element in range
template <class ForwardIterator> ForwardIterator max_element (ForwardIterator first, ForwardIterator last);

82. minmax_element Return smallest and largest elements in range
template <class ForwardIterator> pair<ForwardIterator,ForwardIterator> minmax_element (ForwardIterator first, ForwardIterator last);

83. lexicographical_compare Lexicographical less-than comparison
template <class InputIterator1, class InputIterator2> bool lexicographical_compare (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, InputIterator2 last2);

84. next_permutation Transform range to next permutation
template <class BidirectionalIterator> bool next_permutation (BidirectionalIterator first, BidirectionalIterator last);

85. prev_permutation Transform range to previous permutation
template <class BidirectionalIterator> bool prev_permutation (BidirectionalIterator first, BidirectionalIterator last);

<vector>

1. (constructor) Construct vector
explicit vector(const allocator_type& alloc = allocator_type());
explicit vector(size_type n);
vector(size_type n, const value_type& val, const allocator_type& alloc = allocator_type());
template <class InputIterator> vector(InputIterator first, InputIterator last, const allocator_type& alloc = allocator_type());
vector(const vector& x);
vector(const vector& x, const allocator_type& alloc);
vector(vector&& x);
vector(vector&& x, const allocator_type& alloc);
vector(initializer_list<value_type> il, const allocator_type& alloc = allocator_type());
explicit vector(const allocator_type& alloc = allocator_type());
2. (destructor) Vector destructor
~vector();
3. operator= Assign content
vector& operator=(const vector& x);
vector& operator=(vector&& x);
vector& operator=(initializer_list<value_type> il);
vector& operator=(const vector& x);
4. begin Return iterator to beginning
iterator begin() noexcept;
const_iterator begin() const noexcept;
5. end Return iterator to end
iterator end() noexcept;
const_iterator end() const noexcept;
6. rbegin Return reverse iterator to reverse beginning
reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;
7. rend Return reverse iterator to reverse end
reverse_iterator rend() noexcept;
const_reverse_iterator rend() const noexcept;
8. cbegin Return const_iterator to beginning
const_iterator cbegin() const noexcept;
9. cend Return const_iterator to end
const_iterator cend() const noexcept;
10. crbegin Return const_reverse_iterator to reverse beginning
const_reverse_iterator crbegin() const noexcept;
11. crend Return const_reverse_iterator to reverse end
const_reverse_iterator crend() const noexcept;
12. size Return size
size_type size() const noexcept;
13. max_size Return maximum size
size_type max_size() const noexcept;
14. resize Change size
void resize(size_type n);
void resize(size_type n, const value_type& val);
15. capacity Return size of allocated storage capacity
size_type capacity() const noexcept;
16. empty Test whether vector is empty
bool empty() const noexcept;
17. reserve Request a change in capacity
void reserve (size_type n);
18. shrink_to_fit Shrink to fit
void shrink_to_fit();
19. operator[] Access element
reference operator[] (size_type n); const_reference operator[] (size_type n) const;
20. at Access element
reference at (size_type n); const_reference at (size_type n) const;
21. front Access first element
reference front(); const_reference front() const;
22. back Access last element
reference back(); const_reference back() const;
23. data Access data
value_type* data() noexcept; const value_type* data() const noexcept;
24. assign Assign vector content
template <class InputIterator> void assign(InputIterator first, InputIterator last);

void assign(size_type n, const value_type& val);
void assign(initializer_list<value_type> il);
template <class InputIterator> void assign(InputIterator first, InputIterator last);
25. push_back Add element at the end
void push_back(const value_type& val);
void push_back(value_type&& val);
26. pop_back Delete last element
void pop_back();
27. insert Insert elements
iterator insert(const_iterator position, const value_type& val);
iterator insert(const_iterator position, size_type n, const value_type& val);
template <class InputIterator> iterator insert(const_iterator position, InputIterator first, InputIterator last);
iterator insert(const_iterator position, value_type&& val);
iterator insert(const_iterator position, initializer_list<value_type> il);
iterator insert(const_iterator position, const value_type& val);
28. erase Erase elements
iterator erase(const_iterator position);
iterator erase(const_iterator first, const_iterator last);
29. swap Swap content
void swap (vector& x);
30. clear Clear content
void clear() noexcept;
31. emplace Construct and insert element
template <class... Args> iterator emplace (const_iterator position, Args&&... args);
32. emplace_back Construct and insert element at the end
template <class... Args> void emplace_back (Args&&... args);
33. get_allocator Get allocator
allocator_type get_allocator() const noexcept;
34. relational operators Relational operators for vector
template <class T, class Alloc> bool operator== (const vector<T,Alloc>& lhs, const vector<T,Alloc>& rhs);
35. swap Exchange contents of vectors
template <class T, class Alloc> void swap (vector<T,Alloc>& x, vector<T,Alloc>& y);
36. vector<bool> Vector of bool
template < class T, class Alloc = allocator<T> > class vector; // generic
template template <class Alloc> class vector<bool,Alloc>; // bool specialization

<list>

1. (constructor) Construct list
explicit list(const allocator_type& alloc = allocator_type());
explicit list(size_type n);
list(size_type n, const value_type& val, const allocator_type& alloc = allocator_type());
template <class InputIterator> list(InputIterator first, InputIterator last, const allocator_type& alloc = allocator_type());
list(const list& x);
list(const list& x, const allocator_type& alloc);
list(list&& x);
list(list&& x, const allocator_type& alloc);
list(initializer_list<value_type> il, const allocator_type& alloc = allocator_type());
explicit list(const allocator_type& alloc = allocator_type());
2. (destructor) List destructor
~list();
3. operator= Assign content
list& operator=(const list& x);
list& operator=(list&& x);
list& operator=(initializer_list<value_type> il);
list& operator=(const list& x);
4. begin Return iterator to beginning
iterator begin() noexcept;
const_iterator begin() const noexcept;
5. end Return iterator to end
iterator end() noexcept;
const_iterator end() const noexcept;
6. rbegin Return reverse iterator to reverse beginning
reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;
7. rend Return reverse iterator to reverse end
reverse_iterator rend() nothrow;
const_reverse_iterator rend() const nothrow;
8. cbegin Return const_iterator to beginning
const_iterator cbegin() const noexcept;
9. cend Return const_iterator to end
const_iterator cend() const noexcept;
10. crbegin Return const_reverse_iterator to reverse beginning
const_reverse_iterator crbegin() const noexcept;
11. crend Return const_reverse_iterator to reverse end
const_reverse_iterator crend() const noexcept;
12. empty Test whether container is empty
bool empty() const noexcept;
13. size Return size
size_type size() const noexcept;
14. max_size Return maximum size
size_type max_size() const noexcept;
15. front Access first element
reference front(); const_reference front() const;
16. back Access last element
reference back(); const_reference back() const;
17. assign Assign new content to container
template <class InputIterator> void assign(InputIterator first, InputIterator last);
void assign(size_type n, const value_type& val);
void assign(initializer_list<value_type> il);
template <class InputIterator> void assign(InputIterator first, InputIterator last);
18. emplace_front Construct and insert element at beginning
template <class... Args> void emplace_front (Args&&... args);
19. push_front Insert element at beginning
void push_front(const value_type& val);
void push_front(value_type&& val);
20. pop_front Delete first element
void pop_front();
21. emplace_back Construct and insert element at the end
template <class... Args> void emplace_back (Args&&... args);
22. push_back Add element at the end
void push_back(const value_type& val);
void push_back(value_type&& val);
23. pop_back Delete last element
void pop_back();
24. emplace Construct and insert element
template <class... Args> iterator emplace (const_iterator position, Args&&... args);
25. insert Insert elements
iterator insert(const_iterator position, const value_type& val);
iterator insert(const_iterator position, size_type n, const value_type& val);
template <class InputIterator> iterator insert(const_iterator position, InputIterator first, InputIterator last);
iterator insert(const_iterator position, value_type&& val);
iterator insert(const_iterator position, initializer_list<value_type> il);
iterator insert(const_iterator position, const value_type& val);
26. erase Erase elements
iterator erase(const_iterator position);
iterator erase(const_iterator first, const_iterator last);
27. swap Swap content
void swap (list& x);
28. resize Change size
void resize(size_type n);
void resize(size_type n, const value_type& val);
29. clear Clear content
void clear() noexcept;
30. splice Transfer elements from list to list
void splice(const_iterator position, list& x);
void splice(const_iterator position, list&& x);
void splice(const_iterator position, list& x, const_iterator i);
void splice(const_iterator position, list&& x, const_iterator i);
void splice(const_iterator position, list& x, const_iterator first, const_iterator last);
void splice(const_iterator position, list&& x, const_iterator first, const_iterator last);
void splice(const_iterator position, list& x);
void splice(const_iterator position, list&& x);
31. remove Remove elements with specific value
void remove (const value_type& val);
32. remove_if Remove elements fulfilling condition
template <class Predicate> void remove_if (Predicate pred);
33. unique Remove duplicate values
void unique();
34. merge Merge sorted lists
void merge(list& x);
void merge(list&& x);
template <class Compare> void merge(list& x, Compare comp);
template <class Compare> void merge(list&& x, Compare comp);
void merge(list& x);
void merge(list&& x);
35. sort Sort elements in container

void sort();
36. reverse Reverse the order of elements
void reverse() noexcept;
37. get_allocator Get allocator
allocator_type get_allocator() const noexcept;
38. relational operators (forward_list) Relational operators for forward_list
template <class T, class Alloc> bool operator== (const forward_list<T,Alloc>& lhs, const forward_list<T,Alloc>& rhs);
39. swap (forward_list) Exchanges the contents of two forward_list containers
template <class T, class Alloc> void swap (forward_list<T,Alloc>& x, forward_list<T,Alloc>& y);

<deque>

1. (constructor) Construct deque container
explicit deque(const allocator_type& alloc = allocator_type());
explicit deque(size_type n);
deque(size_type n, const value_type& val, const allocator_type& alloc = allocator_type());
template <class InputIterator> deque(InputIterator first, InputIterator last, const allocator_type& alloc = allocator_type());
deque(const deque& x);
deque(const deque& x, const allocator_type& alloc);
deque(deque&& x);
deque(deque&& x, const allocator_type& alloc);
deque(initializer_list<value_type> il, const allocator_type& alloc = allocator_type());
explicit deque(const allocator_type& alloc = allocator_type());

2. (destructor) Deque destructor
~deque();

3. operator= Assign content
deque& operator=(const deque& x);
deque& operator=(deque&& x);
deque& operator=(initializer_list<value_type> il);
deque& operator=(const deque& x);

4. begin Return iterator to beginning
iterator begin() noexcept;
const_iterator begin() const noexcept;

5. end Return iterator to end
iterator end() noexcept;
const_iterator end() const noexcept;

6. rbegin Return reverse iterator to reverse beginning
reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;

7. rend Return reverse iterator to reverse end
reverse_iterator rend() noexcept;
const_reverse_iterator rend() const noexcept;

8. cbegin Return const_iterator to beginning
const_iterator cbegin() const noexcept;

9. cend Return const_iterator to end
const_iterator cend() const noexcept;

10. crbegin Return const_reverse_iterator to reverse beginning
const_reverse_iterator crbegin() const noexcept;

11. crend Return const_reverse_iterator to reverse end
const_reverse_iterator crend() const noexcept;

12. size Return size
size_type size() const noexcept;

13. max_size Return maximum size
size_type max_size() const noexcept;

14. resize Change size
void resize(size_type n);
void resize(size_type n, const value_type& val);

15. empty Test whether container is empty
bool empty() const noexcept;

16. shrink_to_fit Shrink to fit
void shrink_to_fit();

17. operator[] Access element
reference operator[] (size_type n); const_reference operator[] (size_type n) const;

18. at Access element
reference at (size_type n); const_reference at (size_type n) const;

19. front Access first element
reference front(); const_reference front() const;

20. back Access last element
reference back(); const_reference back() const;

21. assign Assign container content
template <class InputIterator> void assign(InputIterator first, InputIterator last);
void assign(size_type n, const value_type& val);
void assign(initializer_list<value_type> il);
template <class InputIterator> void assign(InputIterator first, InputIterator last);

22. push_back Add element at the end
void push_back(const value_type& val);
void push_back(value_type&& val);

23. push_front Insert element at beginning
void push_front(const value_type& val);
void push_front(value_type&& val);

24. pop_back Delete last element
void pop_back();

25. pop_front Delete first element
void pop_front();

26. insert Insert elements
iterator insert(const_iterator position, const value_type& val);
iterator insert(const_iterator position, size_type n, const value_type& val);
template <class InputIterator> iterator insert(const_iterator position, InputIterator first, InputIterator last);
iterator insert(const_iterator position, value_type&& val);
iterator insert(const_iterator position, initializer_list<value_type> il);
iterator insert(const_iterator position, const value_type& val);

27. erase Erase elements
iterator erase(const_iterator position);
iterator erase(const_iterator first, const_iterator last);

28. swap Swap content
void swap (deque& x);

29. clear Clear content
void clear() noexcept;

30. emplace Construct and insert element
template <class... Args> iterator emplace (const_iterator position, Args&&... args);

31. emplace_front Construct and insert element at beginning
template <class... Args> void emplace_front (Args&&... args);

32. emplace_back Construct and insert element at the end
template <class... Args> void emplace_back (Args&&... args);

33. get_allocator Get allocator
allocator_type get_allocator() const noexcept;

34. relational operators Relational operators for deque
template <class T, class Alloc> bool operator== (const deque<T,Alloc>& lhs, const deque<T,Alloc>& rhs);

35. swap Exchanges the contents of two deque containers
template <class T, class Alloc> void swap (deque<T,Alloc>& x, deque<T,Alloc>& y);

<queue>

1. (constructor) Construct queue
explicit queue(const container_type& ctnr);
explicit queue(container_type&& ctnr = container_type());
template <class Alloc> explicit queue(const Alloc& alloc);
template <class Alloc> queue(const container_type& ctnr, const Alloc& alloc);
template <class Alloc> queue(container_type&& ctnr, const Alloc& alloc);
template <class Alloc> queue(const queue& x, const Alloc& alloc);
template <class Alloc> queue(queue&& x, const Alloc& alloc);
explicit queue(const container_type& ctnr);

2. empty Test whether container is empty
bool empty() const;

3. size Return size
size_type size() const;

4. front Access next element
reference& front();
const_reference& front() const;

5. back Access last element
reference& back();
const_reference& back() const;

6. push Insert element
void push(const value_type& val);
void push(value_type&& val);

7. emplace Construct and insert element
template <class... Args> void emplace (Args&&... args);

8. pop Remove next element
void pop();

9. swap Swap contents
void swap (queue& x) noexcept(/*see below*/);

10. relational operators Relational operators for queue
template <class T, class Container> bool operator== (const queue<T,Container>& lhs, const queue<T,Container>& rhs);

11. swap (queue) Exchange contents of queues
template <class T, class Container> void swap (queue<T,Container>& x, queue<T,Container>& y) noexcept(noexcept(x.swap(y)));

12. uses_allocator<queue> Uses allocator for queue
template <class T, class Container, class Alloc> struct uses_allocator<queue<T,Container>,Alloc>;

<stack>

1. (constructor) Construct stack
explicit stack(const container_type& ctnr);
explicit stack(container_type&& ctnr = container_type());
template <class Alloc> explicit stack(const Alloc& alloc);
template <class Alloc> stack(const container_type& ctnr, const Alloc& alloc);
template <class Alloc> stack(container_type&& ctnr, const Alloc& alloc);
template <class Alloc> stack(const stack& x, const Alloc& alloc);
template <class Alloc> stack(stack&& x, const Alloc& alloc);
explicit stack(const container_type& ctnr);
2. empty Test whether container is empty
bool empty() const;
3. size Return size
size_type size() const;
4. top Access next element
reference& top();
const_reference& top() const;
5. push Insert element
void push(const value_type& val);
void push(value_type&& val);
6. emplace Construct and insert element
template <class... Args> void emplace (Args&&... args);
7. pop Remove top element
void pop();
8. swap Swap contents
void swap (stack& x) noexcept(/*see below*/);
9. relational operators Relational operators for stack
template <class T, class Container> bool operator== (const stack<T,Container>& lhs, const stack<T,Container>& rhs);
10. swap (stack) Exchange contents of stacks
template <class T, class Container> void swap (stack<T,Container>& x, stack<T,Container>& y) noexcept(noexcept(x.swap(y)));
11. uses_allocator<stack> Uses allocator for stack
template <class T, class Container, class Alloc> struct uses_allocator<stack<T,Container>,Alloc>;

<set>

(constructor) Construct multiset
explicit multiset(const key_compare& comp = key_compare(), const allocator_type& alloc = allocator_type());
explicit multiset(const allocator_type& alloc);
template <class InputIterator> multiset(InputIterator first, InputIterator last, const key_compare& comp = key_compare(), const allocator_type& = allocator_type());
multiset(const multiset& x);
multiset(const multiset& x, const allocator_type& alloc);
multiset(multiset&& x);
multiset(multiset&& x, const allocator_type& alloc);
multiset(initializer_list<value_type> il, const key_compare& comp = key_compare(), const allocator_type& alloc = allocator_type());
explicit multiset(const key_compare& comp = key_compare(), const allocator_type& alloc = allocator_type());
explicit multiset(const allocator_type& alloc);
2. (destructor) Multiset destructor
~multiset();
3. operator= Copy container content
multiset& operator=(const multiset& x);
multiset& operator=(multiset&& x);
multiset& operator=(initializer_list<value_type> il);
multiset& operator=(const multiset& x);
4. begin Return iterator to beginning
iterator begin() noexcept;
const_iterator begin() const noexcept;
5. end Return iterator to end
iterator end() noexcept;
const_iterator end() const noexcept;
6. rbegin Return reverse iterator to reverse beginning
reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;
7. rend Return reverse iterator to reverse end
reverse_iterator rend() nothrow;
const_reverse_iterator rend() const nothrow;
8. cbegin Return const_iterator to beginning
const_iterator cbegin() const noexcept;
9. cend Return const_iterator to end
const_iterator cend() const noexcept;
10. crbegin Return const_reverse_iterator to reverse beginning
const_reverse_iterator crbegin() const noexcept;
11. crend Return const_reverse_iterator to reverse end
const_reverse_iterator crend() const noexcept;
12. empty Test whether container is empty
bool empty() const noexcept;
13. size Return container size
size_type size() const noexcept;
14. max_size Return maximum size
size_type max_size() const noexcept;
15. insert Insert element
iterator insert(const value_type& val);
iterator insert(value_type&& val);
iterator insert(const_iterator position, const value_type& val);
iterator insert(const_iterator position, value_type&& val);
template <class InputIterator> void insert(InputIterator first, InputIterator last);
void insert(initializer_list<value_type> il);
iterator insert(const value_type& val);
iterator insert(value_type&& val);
16. erase Erase elements
iterator erase(const_iterator position);
size_type erase(const value_type& val);
iterator erase(const_iterator first, const_iterator last);
iterator erase(const_iterator position);
17. swap Swap content
void swap (multiset& x);
18. clear Clear content
void clear() noexcept;
19. emplace Construct and insert element
template <class... Args> iterator emplace (Args&&... args);
20. emplace_hint Construct and insert element with hint
template <class... Args> iterator emplace_hint (const_iterator position,

Args&&... args);
21. key_comp Return comparison object
key_compare key_comp() const;
22. value_comp Return comparison object
value_compare value_comp() const;
23. find Get iterator to element
const_iterator find(const value_type& val) const;
iterator find(const value_type& val);
24. count Count elements with a specific key
size_type count (const value_type& val) const;
25. lower_bound Return iterator to lower bound
const_iterator lower_bound(const value_type& val) const;
iterator lower_bound(const value_type& val);
26. upper_bound Return iterator to upper bound
const_iterator upper_bound(const value_type& val) const;
iterator upper_bound(const value_type& val);
27. equal_range Get range of equal elements
pair<const_iterator,const_iterator> equal_range(const value_type& val) const;
pair<iterator,iterator> equal_range(const value_type& val);
28. get_allocator Get allocator
allocator_type get_allocator() const noexcept;

<unordered_set>

1. (constructor) Construct unordered_set
explicit unordered_set(size_type n = /* see below */, const hasher& hf = hasher(), const key_equal& eql = key_equal(), const allocator_type& alloc = allocator_type());
explicit unordered_set(const allocator_type& alloc);
template <class InputIterator> unordered_set(InputIterator first, InputIterator last, size_type n = /* see below */, const hasher& hf = hasher(), const key_equal& eql = key_equal(), const allocator_type& alloc = allocator_type());
unordered_set(const unordered_set& ust);
unordered_set(const unordered_set& ust, const allocator_type& alloc);
unordered_set(unordered_set&& ust);
unordered_set(unordered_set&& ust, const allocator_type& alloc);
unordered_set(initializer_list<value_type> il, size_type n = /* see below */, const hasher& hf = hasher(), const key_equal& eql = key_equal(), const allocator_type& alloc = allocator_type());
explicit unordered_set(size_type n = /* see below */, const hasher& hf = hasher(), const key_equal& eql = key_equal(), const allocator_type& alloc = allocator_type());
explicit unordered_set(const allocator_type& alloc);

2. (destructor) Destroy unordered set
~unordered_set();

3. operator= Assign content
unordered_set& operator= (const unordered_set& ust);

4. empty Test whether container is empty
bool empty() const noexcept;

5. size Return container size
size_type size() const noexcept;

6. max_size Return maximum size
size_type max_size() const noexcept;

7. begin Return iterator to beginning
iterator begin() noexcept; const_iterator begin() const noexcept;

8. end Return iterator to end
iterator end() noexcept; const_iterator end() const noexcept;

9. cbegin Return const_iterator to beginning
const_iterator cbegin() const noexcept;

10. cend Return const_iterator to end
const_iterator cend() const noexcept;

11. find Get iterator to element
iterator find (const key_type& k); const_iterator find (const key_type& k) const;

12. count Count elements with a specific key
size_type count (const key_type& k) const;

13. equal_range Get range of elements with a specific key
pair<iterator,iterator> equal_range (const key_type& k);
pair<const_iterator,const_iterator> equal_range (const key_type& k) const;

14. emplace Construct and insert element
template <class... Args> pair <iterator,bool> emplace (Args&&... args);

15. emplace_hint Construct and insert element with hint
template <class... Args> iterator emplace_hint (const_iterator position, Args&&... args);

16. insert Insert elements
pair<iterator,bool> insert (const value_type& val);

17. erase Erase elements
iterator erase (const_iterator position);

18. clear Clear content
void clear() noexcept;

19. swap Swap content
void swap (unordered_set& ust);

20. bucket_count Return number of buckets
size_type bucket_count() const noexcept;

21. max_bucket_count Return maximum number of buckets
size_type max_bucket_count() const noexcept;

22. bucket_size Return bucket size
size_type bucket_size (size_type n) const;

23. bucket Locate element's bucket
size_type bucket (const key_type& k) const;

24. load_factor Return load factor
float load_factor() const noexcept;

25. max_load_factor Get or set maximum load factor
float max_load_factor() const noexcept;

26. rehash Set number of buckets
void rehash (size_type n);

27. reserve Request a capacity change
void reserve (size_type n);

28. hash_function Get hash function
hasher hash_function() const;

29. key_eq Get key equivalence predicate
key_equal key_eq() const;

30. get_allocator Get allocator
allocator_type get_allocator() const noexcept;

31. operators (unordered_set) Relational operators for
template <class Key, class Hash, class Pred, class Alloc> bool operator== (const unordered_set<Key,Hash,Pred,Alloc>& lhs, const unordered_set<Key,Hash,Pred,Alloc>& rhs);

32. swap (unordered_set) Exchanges contents of two unordered_set containers
template <class Key, class Hash, class Pred, class Alloc> void swap (unordered_set<Key,Hash,Pred,Alloc>& lhs, unordered_set<Key,Hash,Pred,Alloc>& rhs);

<map>

1. (constructor) Construct map
explicit map(const key_compare& comp = key_compare(), const allocator_type& alloc = allocator_type());
explicit map(const allocator_type& alloc);
template <class InputIterator> map(InputIterator first, InputIterator last, const key_compare& comp = key_compare(), const allocator_type& = allocator_type());
map(const map& x);
map(const map& x, const allocator_type& alloc);
map(map&& x);
map(map&& x, const allocator_type& alloc);
map(initializer_list<value_type> il, const key_compare& comp = key_compare(), const allocator_type& alloc = allocator_type());

explicit map(const key_compare& comp = key_compare(), const allocator_type& alloc = allocator_type());
explicit map(const allocator_type& alloc);

2. (destructor) Map destructor
~map();

3. operator= Copy container content
map& operator=(const map& x);
map& operator=(map&& x);
map& operator=(initializer_list<value_type> il);
map& operator=(const map& x);

4. begin Return iterator to beginning
iterator begin() noexcept;
const_iterator begin() const noexcept;

5. end Return iterator to end
iterator end() noexcept;
const_iterator end() const noexcept;

6. rbegin Return reverse iterator to reverse beginning
reverse_iterator rbegin() noexcept;
const_reverse_iterator rbegin() const noexcept;

7. rend Return reverse iterator to reverse end
reverse_iterator rend() noexcept;
const_reverse_iterator rend() const noexcept;

8. cbegin Return const_iterator to beginning
const_iterator cbegin() const noexcept;

9. cend Return const_iterator to end
const_iterator cend() const noexcept;

10. crbegin Return const_reverse_iterator to reverse beginning
const_reverse_iterator crbegin() const noexcept;

11. crend Return const_reverse_iterator to reverse end
const_reverse_iterator crend() const noexcept;

12. empty Test whether container is empty
bool empty() const noexcept;

13. size Return container size
size_type size() const noexcept;

14. max_size Return maximum size
size_type max_size() const noexcept;

15. operator[] Access element
mapped_type& operator[](const key_type& k);
mapped_type& operator[](key_type&& k);

16. at Access element
mapped_type& at (const key_type& k); const mapped_type& at (const key_type& k) const;

17. insert Insert elements
pair<iterator,bool> insert(const value_type& val);
template <class P> pair<iterator,bool> insert(P&& val);
iterator insert(const_iterator position, const value_type& val);
template <class P> iterator insert(const_iterator position, P&& val);
template <class InputIterator> void insert(InputIterator first, InputIterator last);
void insert(initializer_list<value_type> il);
pair<iterator,bool> insert(const value_type& val);
template <class P> pair<iterator,bool> insert(P&& val);

18. erase Erase elements
iterator erase(const_iterator position);
size_type erase(const key_type& k);
iterator erase(const_iterator first, const_iterator last);
iterator erase(const_iterator position);

19. swap Swap content
void swap (map& x);

20. clear Clear content
void clear() noexcept;

21. emplace Construct and insert element
template <class... Args> pair<iterator,bool> emplace (Args&&... args);

22. emplace_hint Construct and insert element with hint
template <class... Args> iterator emplace_hint (const_iterator position, Args&&... args);

23. key_comp Return key comparison object
key_compare key_comp() const;

24. value_comp Return value comparison object
value_compare value_comp() const;

25. find Get iterator to element
iterator find (const key_type& k); const_iterator find (const key_type& k) const;

26. count Count elements with a specific key
size_type count (const key_type& k) const;

27. lower_bound Return iterator to lower bound
iterator lower_bound (const key_type& k); const_iterator lower_bound (const key_type& k) const;

28. upper_bound Return iterator to upper bound
iterator upper_bound (const key_type& k); const_iterator upper_bound (const key_type& k) const;

29. equal_range Get range of equal elements
pair<const_iterator,const_iterator> equal_range (const key_type& k) const;
pair<iterator,iterator> equal_range (const key_type& k);

30. get_allocator Get allocator
allocator_type get_allocator() const noexcept;

<unordered_map>

1. (constructor) Construct unordered_map
explicit unordered_map(size_type n = /* see below */, const hasher& hf = hasher(), const key_equal& eql = key_equal(), const allocator_type& alloc = allocator_type());
explicit unordered_map(const allocator_type& alloc);
template <class InputIterator> unordered_map(InputIterator first, InputIterator last, size_type n = /* see below */, const hasher& hf = hasher(), const key_equal& eql = key_equal(), const allocator_type& alloc = allocator_type());
unordered_map(const unordered_map& ump);
unordered_map(const unordered_map& ump, const allocator_type& alloc);
unordered_map(unordered_map&& ump);
unordered_map(unordered_map&& ump, const allocator_type& alloc);
unordered_map(initializer_list<value_type> il, size_type n = /* see below */, const hasher& hf = hasher(), const key_equal& eql = key_equal(), const allocator_type& alloc = allocator_type());
explicit unordered_map(size_type n = /* see below */, const hasher& hf = hasher(), const key_equal& eql = key_equal(), const allocator_type& alloc = allocator_type());
explicit unordered_map(const allocator_type& alloc);
2. (destructor) Destroy unordered map
~unordered_map();
3. operator= Assign content
unordered_map& operator= (const unordered_map& ump);
4. empty Test whether container is empty
bool empty() const noexcept;
5. size Return container size
size_type size() const noexcept;
6. max_size Return maximum size
size_type max_size() const noexcept;
7. begin Return iterator to beginning
iterator begin() noexcept; const_iterator begin() const noexcept;
8. end Return iterator to end
iterator end() noexcept; const_iterator end() const noexcept;
9. cbegin Return const_iterator to beginning
const_iterator cbegin() const noexcept;
10. cend Return const_iterator to end
const_iterator cend() const noexcept;
11. operator[] Access element
mapped_type& operator[] (const key_type& k); mapped_type& operator[] (key_type&& k);
12. at Access element
mapped_type& at (const key_type& k); const mapped_type& at (const key_type& k) const;

13. find Get iterator to element
iterator find (const key_type& k); const_iterator find (const key_type& k) const;
14. count Count elements with a specific key
size_type count (const key_type& k) const;
15. equal_range Get range of elements with specific key
pair<iterator,iterator> equal_range (const key_type& k);
pair<const_iterator,const_iterator> equal_range (const key_type& k) const;
16. emplace Construct and insert element
template <class... Args> pair<iterator, bool> emplace (Args&&... args);
17. emplace_hint Construct and insert element with hint
template <class... Args> iterator emplace_hint (const_iterator position, Args&&... args);
18. insert Insert elements
pair<iterator,bool> insert (const value_type& val);
19. erase Erase elements
iterator erase (const_iterator position);
20. clear Clear content
void clear() noexcept;
21. swap Swap content
void swap (unordered_map& ump);
22. bucket_count Return number of buckets
size_type bucket_count() const noexcept;
23. max_bucket_count Return maximum number of buckets
size_type max_bucket_count() const noexcept;
24. bucket_size Return bucket size
size_type bucket_size (size_type n) const;
25. bucket Locate element's bucket
size_type bucket (const key_type& k) const;
26. load_factor Return load factor
float load_factor() const noexcept;
27. max_load_factor Get or set maximum load factor
float max_load_factor() const noexcept;
28. rehash Set number of buckets
void rehash(size_type n);
29. reserve Request a capacity change
void reserve (size_type n);
30. hash_function Get hash function
hasher hash_function() const;
31. key_eq Get key equivalence predicate
key_equal key_eq() const;
32. get_allocator Get allocator
allocator_type get_allocator() const noexcept;
33. operators (unordered_map) Relational operators for unordered_map
template <class Key, class T, class Hash, class Pred, class Alloc> bool operator== (const unordered_map<Key,T,Hash,Pred,Alloc>& lhs, const unordered_map<Key,T,Hash,Pred,Alloc>& rhs);
34. swap (unordered_map) Exchanges contents of two unordered_map containers
template <class Key, class T, class Hash, class Pred, class Alloc> void swap (unordered_map<Key,T,Hash,Pred,Alloc>& lhs, unordered_map<Key,T,Hash,Pred,Alloc>& rhs);

<iterator>

1. advance Advance iterator
template <class InputIterator, class Distance> void advance (InputIterator& it, Distance n);

2. distance Return distance between iterators
template<class InputIterator> typename iterator_traits<InputIterator>::difference_type distance (InputIterator first, InputIterator last);

3. begin Iterator to beginning
template <class Container> auto begin(Container& cont) -> decltype(cont.begin());
template <class Container> auto begin(const Container& cont) -> decltype(cont.begin());
template <class T, size_t N> T* begin(T(&arr)[N]);
template <class Container> auto begin(Container& cont) -> decltype(cont.begin());
template <class Container> auto begin(const Container& cont) -> decltype(cont.begin());

4. end Iterator to end
template <class Container> auto end(Container& cont) -> decltype(cont.end());
template <class Container> auto end(const Container& cont) -> decltype(cont.end());
template <class T, size_t N> T* end(T(&arr)[N]);
template <class Container> auto end(Container& cont) -> decltype(cont.end());
template <class Container> auto end(const Container& cont) -> decltype(cont.end());

5. prev Get iterator to previous element
template <class BidirectionalIterator> BidirectionalIterator prev (BidirectionalIterator it, typename iterator_traits<BidirectionalIterator>::difference_type n = 1);

6. next Get iterator to next element
template <class ForwardIterator> ForwardIterator next (ForwardIterator it, typename iterator_traits<ForwardIterator>::difference_type n = 1);

7. back_inserter Construct back insert iterator
template <class Container> back_insert_iterator<Container> back_inserter (Container& x);

8. front_inserter Constructs front insert iterator
template <class Container> front_insert_iterator<Container> front_inserter (Container& x);

9. inserter Construct insert iterator
template <class Container> insert_iterator<Container> inserter(Container& x, typename Container::iterator it);

10. make_move_iterator Construct move iterator
template <class Iterator> move_iterator<Iterator> make_move_iterator(const Iterator& it);

11. iterator Iterator base class
template <class Category, // iterator::iterator_category class T, // iterator::value_type class Distance = ptrdiff_t, // iterator::difference_type class Pointer = T*, // iterator::pointer class Reference = T& // iterator::reference > class iterator;

12. iterator_traits Iterator traits
template <class Iterator> class iterator_traits; template <class T> class iterator_traits<T*>; template <class T> class iterator_traits<const T*>;

13. reverse_iterator Reverse iterator
template <class Iterator> class reverse_iterator;

14. move_iterator Move iterator
template <class Iterator> class move_iterator;

15. back_insert_iterator Back insert iterator

16. front_insert_iterator Front insert iterator

17. insert_iterator Insert iterator

18. istream_iterator Istream iterator
template <class T, class charT=char, class traits=char_traits<charT>, class Distance = ptrdiff_t> class istream_iterator;

19. ostream_iterator Ostream iterator
template <class T, class charT=char, class traits=char_traits<charT> > class ostream_iterator;

20. istreambuf_iterator Input stream buffer iterator
template <class charT, class traits=char_traits<charT> > class istreambuf_iterator;

21. ostreambuf_iterator Output stream buffer iterator
template <class charT, class traits=char_traits<charT> > class ostreambuf_iterator;

22. input_iterator_tag Input iterator category
struct input_iterator_tag {};

23. output_iterator_tag Output iterator category
struct output_iterator_tag {};

24. forward_iterator_tag Forward iterator category
struct forward_iterator_tag {};

25. bidirectional_iterator_tag Bidirectional iterator category
struct bidirectional_iterator_tag {}

26. random_access_iterator_tag Random-access iterator category
struct random_access_iterator_tag {};

<bitset>

1. reference Reference-like type
class bitset::reference { friend class bitset;
reference() noexcept;
// no public constructor public: ~reference();
operator bool() const noexcept;
// convert to bool reference& operator=(bool x) noexcept;
// assign bool reference& operator=(const reference& x) noexcept;
// assign bit reference& flip() noexcept;
// flip bit value bool operator~() const noexcept;
// return inverse value };
2. (constructor) Construct bitset
constexpr bitset() noexcept;
constexpr bitset(unsigned long long val) noexcept;
template <class charT, class traits, class Alloc> explicit bitset(const basic_string<charT,traits,Alloc>& str, typename basic_string<charT,traits,Alloc>::size_type pos = 0, typename basic_string<charT,traits,Alloc>::size_type n = basic_string<charT,traits,Alloc>::npos, charT zero = charT('0'), charT one = charT('1'));
template <class charT> explicit bitset(const charT* str, typename basic_string<charT>::size_type n = basic_string<charT>::npos, charT zero = charT('0'), charT one = charT('1'));
constexpr bitset() noexcept;
3. applicable operators Bitset operators
bitset& operator&=(const bitset& rhs) noexcept;
bitset& operator|=(const bitset& rhs) noexcept;
bitset& operator^=(const bitset& rhs) noexcept;
bitset& operator<<=(size_t pos) noexcept;
bitset& operator>>=(size_t pos) noexcept;
bitset operator~() const noexcept;
bitset operator<<(size_t pos) const noexcept;
bitset operator>>(size_t pos) const noexcept;
bool operator==(const bitset& rhs) const noexcept;
bool operator!=(const bitset& rhs) const noexcept;
template<size_t N> bitset<N> operator&(const bitset<N>& lhs, const bitset<N>& rhs) noexcept;
template<size_t N> bitset<N> operator|(const bitset<N>& lhs, const bitset<N>& rhs) noexcept;
template<size_t N> bitset<N> operator^(const bitset<N>& lhs, const bitset<N>& rhs) noexcept;
template<class charT, class traits, size_t N> basic_istream<charT, traits>& operator>>(basic_istream<charT,traits>& is, bitset<N>& rhs);
template<class charT, class traits, size_t N> basic_ostream<charT, traits>& operator<<(basic_ostream<charT,traits>& os, const bitset<N>& rhs);
bitset& operator&=(const bitset& rhs) noexcept;
bitset& operator|=(const bitset& rhs) noexcept;
bitset& operator^=(const bitset& rhs) noexcept;
bitset& operator<<=(size_t pos) noexcept;
bitset& operator>>=(size_t pos) noexcept;
bitset operator~() const noexcept;
bitset operator<<(size_t pos) const noexcept;
bitset operator>>(size_t pos) const noexcept;
bool operator==(const bitset& rhs) const noexcept;
bool operator!=(const bitset& rhs) const noexcept;
4. operator[] Access bit
bool operator[] (size_t pos) const; reference operator[] (size_t pos);
5. count Count bits set
size_t count() const noexcept;
6. size Return size
constexpr size_t size() noexcept;
7. test Return bit value
bool test (size_t pos) const;
8. any Test if any bit is set
bool any() const noexcept;
9. none Test if no bit is set
bool none() const noexcept;
10. all Test if all bits are set
bool all() const noexcept;
11. set Set bits
bitset& set() constexpr;
bitset& set(size_t pos, bool val = true);
bitset& set() constexpr;
12. reset Reset bits
bitset& reset() noexcept;
bitset& reset(size_t pos);
bitset& reset() noexcept;
13. flip Flip bits
bitset& flip() noexcept;
bitset& flip(size_t pos);
bitset& flip() noexcept;
14. to_string Convert to string
template <class charT = char, class traits = char_traits<charT>, class Alloc = allocator<charT>> basic_string<charT,traits,Alloc> to_string(charT zero = charT('0'), charT one = charT('1')) const;
15. to_ulong Convert to unsigned long integer
unsigned long to_ulong() const;
16. to_ullong Convert to unsigned long long
unsigned long long to_ullong() const;
17. applicable operators Bitset operators
bitset& operator&=(const bitset& rhs) noexcept;
bitset& operator|=(const bitset& rhs) noexcept;
bitset& operator^=(const bitset& rhs) noexcept;
bitset& operator<<=(size_t pos) noexcept;
bitset& operator>>=(size_t pos) noexcept;
bitset operator~() const noexcept;
bitset operator<<(size_t pos) const noexcept;
bitset operator>>(size_t pos) const noexcept;
bool operator==(const bitset& rhs) const noexcept;
bool operator!=(const bitset& rhs) const noexcept;
template<size_t N> bitset<N> operator&(const bitset<N>& lhs, const bitset<N>& rhs) noexcept;
template<size_t N> bitset<N> operator|(const bitset<N>& lhs, const bitset<N>& rhs) noexcept;
template<size_t N> bitset<N> operator^(const bitset<N>& lhs, const bitset<N>& rhs) noexcept;
template<class charT, class traits, size_t N> basic_istream<charT, traits>& operator>>(basic_istream<charT,traits>& is, bitset<N>& rhs);
template<class charT, class traits, size_t N> basic_ostream<charT, traits>& operator<<(basic_ostream<charT,traits>& os, const bitset<N>& rhs);
bitset& operator&=(const bitset& rhs) noexcept;
bitset& operator|=(const bitset& rhs) noexcept;
bitset& operator^=(const bitset& rhs) noexcept;
bitset& operator<<=(size_t pos) noexcept;
bitset& operator>>=(size_t pos) noexcept;
bitset operator~() const noexcept;
bitset operator<<(size_t pos) const noexcept;
bitset operator>>(size_t pos) const noexcept;
bool operator==(const bitset& rhs) const noexcept;
bool operator!=(const bitset& rhs) const noexcept;
18. hash<bitset> Hash for bitset
template <class T> struct hash; // unspecialized template <size_t N> struct hash<bitset<N>>; // bitset specialization

<cmath>

1. cos Compute cosine
double cos(double x);
float cos(float x);
long double cos(long double x);
double cos(T x);
2. sin Compute sine
double sin(double x);
float sin(float x);
long double sin(long double x);
double sin(T x);
3. tan Compute tangent
double tan(double x);
float tan(float x);
long double tan(long double x);
double tan(T x);
4. acos Compute arc cosine
double acos(double x);
float acos(float x);
long double acos(long double x);
double acos(T x);
5. asin Compute arc sine
double asin(double x);
float asin(float x);
long double asin(long double x);
double asin(T x);
6. atan Compute arc tangent
double atan(double x);
float atan(float x);
long double atan(long double x);
double atan(T x);

7. atan2 Compute arc tangent with two parameters
double atan2(double y, double x);
float atan2(float y, float x);
long double atan2(long double y, long double x);
double atan2(Type1 y, Type2 x);
8. cosh Compute hyperbolic cosine
double cosh(double x);
float cosh(float x);
long double cosh(long double x);
double cosh(T x);
9. sinh Compute hyperbolic sine
double sinh(double x);
float sinh(float x);
long double sinh(long double x);
double sinh(T x);
10. tanh Compute hyperbolic tangent
double tanh(double x);
float tanh(float x);
long double tanh(long double x);
double tanh(T x);
11. acosh Compute arc hyperbolic cosine
double acosh(double x);
float acosh(float x);
long double acosh(long double x);
double acosh(T x);
12. asinh Compute arc hyperbolic sine
double asinh(double x);
float asinh(float x);
long double asinh(long double x);
double asinh(T x);
13. atanh Compute arc hyperbolic tangent
double atanh(double x);
float atanh(float x);
long double atanh(long double x);
double asinh(T x);
14. exp Compute exponential function
double exp(double x);
float exp(float x);
long double exp(long double x);
double exp(T x);
15. frexp Get significand and exponent
double frexp(double x, int* exp);
float frexp(float x, int* exp);
long double frexp(long double x, int* exp);
double frexp(T x, int* exp);
16. ldexp Generate value from significand and exponent
double ldexp(double x, int exp);
float ldexp(float x, int exp);
long double ldexp(long double x, int exp);
double ldexp(T x, int exp);
17. log Compute natural logarithm
double log(double x);
float log(float x);
long double log(long double x);
double log(T x);
18. log10 Compute common logarithm
double log10(double x);
float log10(float x);
long double log10(long double x);
double log10(T x);
19. modf Break into fractional and integral parts
double modf(double x, double* intpart);
float modf(float x, float* intpart);
long double modf(long double x, long double* intpart);
double modf(T x, double* intpart);
20. exp2 Compute binary exponential function
double exp2(double x);
float exp2(float x);
long double exp2(long double x);
double exp2(T x);
21. expm1 Compute exponential minus one
double expm1(double x);
float expm1(float x);
long double expm1(long double x);
double expm1(T x);
22. ilogb Integer binary logarithm
int ilogb(double x);
int ilogb(float x);
int ilogb(long double x);
int ilogb(T x);
23. log1p Compute logarithm plus one
double log1p(double x);
float log1p(float x);
long double log1p(long double x);
double log1p(T x);
24. log2 Compute binary logarithm
double log2(double x);
float log2(float x);
long double log2(long double x);
double log2(T x);
25. logb Compute floating-point base logarithm
double logb(double x);
float logb(float x);
long double logb(long double x);
double logb(T x);
26. scalbn Scale significand using floating-point base exponent
double scalbn(double x, int n);
float scalbn(float x, int n);
long double scalbn(long double x, int n);
double scalbn(T x, int n);
27. scalbln Scale significand using floating-point base exponent (long)
double scalbln(double x, long int n);
float scalbln(float x, long int n);
long double scalbln(long double x, long int n);
double scalbln(T x, long int n);
28. pow Raise to power
double pow(double base, double exponent);
float pow(float base, float exponent);
long double pow(long double base, long double exponent);
double pow(Type1 base, Type2 exponent);
29. sqrt Compute square root
double sqrt(double x);
float sqrt(float x);
long double sqrt(long double x);
double sqrt(T x);
30. cbrt Compute cubic root
double cbrt(double x);
float cbrt(float x);
long double cbrt(long double x);
double cbrt(T x);
31. hypot Compute hypotenuse
double hypot(double x, double y);
float hypot(float x, float y);
long double hypot(long double x, long double y);
double hypot(Type1 x, Type2 y);
32. erf Compute error function
double erf(double x);
float erf(float x);
long double erf(long double x);
double erf(T x);
33. erfc Compute complementary error function
double erfc(double x);
float erfc(float x);
long double erfc(long double x);float floor(float x);
long double floor(long double x);
double floor(T x);
38. fmod Compute remainder of division
double fmod(double numer, double denom);
float fmod(float numer, float denom);
long double fmod(long double numer, long double denom);
double fmod(Type1 numer, Type2 denom);
39. trunc Truncate value
double trunc(double x);
float trunc(float x);
long double trunc(long double x);
double trunc(T x);
40. round Round to nearest
double round(double x);
float round(float x);
long double round(long double x);
double round(T x);
41. lround Round to nearest and cast to long integer
long int lround(double x);
long int lround(float x);
long int lround(long double x);
long int lround(T x);
42. llround Round to nearest and cast to long long integer
long long int llround(double x);
long long int llround(float x);
long long int llround(long double x);
long long int llround(T x);
43. rint Round to integral value

double rint(double x);
float rint(float x);
long double rint(long double x);
double rint(T x);

44. lrint Round and cast to long integer
long int lrint(double x);
long int lrint(float x);
long int lrint(long double x);
long int lrint(T x);

45. llrint Round and cast to long long integer
long long int llrint(double x);
long long int llrint(float x);
long long int llrint(long double x);
long long int llrint(T x);

46. nearbyint Round to nearby integral value
double nearbyint(double x);
float nearbyint(float x);
long double nearbyint(long double x);
double nearbyint(T x);

47. remainder Compute remainder (IEC 60559)
double remainder(double numer, double denom);
float remainder(float numer, float denom);
long double remainder(long double numer, long double denom);
double remainder(Type1 numer, Type2 denom);

48. remquo Compute remainder and quotient
double remquo(double numer, double denom, int* quot);
float remquo(float numer, float denom, int* quot);
long double remquo(long double numer, long double denom, int* quot);
double remquo(Type1 numer, Type2 denom, int* quot);

49. copysign Copy sign
double copysign(double x, double y);
float copysign(float x, float y);
long double copysign(long double x, long double y);
double copysign(Type1 x, Type2 y);

50. nan Generate quiet NaN
double nan (const char* tagp);

51. nextafter Next representable value
double nextafter(double x, double y);
float nextafter(float x, float y);
long double nextafter(long double x, long double y);
double nextafter(Type1 x, Type2 y);

52. nexttoward Next representable value toward precise value
double nexttoward(double x, long double y);
float nexttoward(float x, long double y);
long double nexttoward(long double x, long double y);
double nexttoward(T x, long double y);

53. fdim Positive difference
double fdim(double x, double y);
float fdim(float x, float y);
long double fdim(long double x, long double y);
double fdim(Type1 x, Type2 y);

54. fmax Maximum value
double fmax(double x, double y);
float fmax(float x, float y);
long double fmax(long double x, long double y);
double fmax(Type1 x, Type2 y);

55. fmin Minimum value
double fmin(double x, double y);
float fmin(float x, float y);
long double fmin(long double x, long double y);
double fmin(Type1 x, Type2 y);

56. fabs Compute absolute value
double fabs(double x);
float fabs(float x);
long double fabs(long double x);
double fabs(T x);

57. abs Compute absolute value
double abs(double x);
float abs(float x);
long double abs(long double x);
double abs(T x);

58. fma Multiply-add
double fma(double x, double y, double z);
float fma(float x, float y, float z);double erfc(T x);

34. tgamma Compute gamma function
double tgamma(double x);
float tgamma(float x);
long double tgamma(long double x);
double tgamma(T x);

35. lgamma Compute log-gamma function
double lgamma(double x);
float lgamma(float x);
long double lgamma(long double x);
double lgamma(T x);

36. ceil Round up value
double ceil(double x);
float ceil(float x);
long double ceil(long double x);
double ceil(T x);

37. floor Round down value
double floor(double x);long double fma(long double x, long double y, long double z);
double fma(Type1 x, Type2 y, Type3 z);

59. fpclassify Classify floating-point value
int fpclassify(float x);
int fpclassify(double x);
int fpclassify(long double x);
int fpclassify(float x);
int fpclassify(double x);
int fpclassify(long double x);

60. isfinite Is finite value
bool isfinite(float x);
bool isfinite(double x);
bool isfinite(long double x);
bool isfinite(float x);
bool isfinite(double x);
bool isfinite(long double x);

61. isinf Is infinity
bool isinf(float x);
bool isinf(double x);
bool isinf(long double x);
bool isinf(float x);
bool isinf(double x);
bool isinf(long double x);

62. isnan Is Not-A-Number
bool isnan(float x);
bool isnan(double x);
bool isnan(long double x);
bool isnan(float x);
bool isnan(double x);
bool isnan(long double x);

63. isnormal Is normal
bool isnormal(float x);
bool isnormal(double x);
bool isnormal(long double x);
bool isnormal(float x);
bool isnormal(double x);
bool isnormal(long double x);

64. signbit Sign bit
bool signbit(float x);
bool signbit(double x);
bool signbit(long double x);
bool signbit(float x);
bool signbit(double x);
bool signbit(long double x);

65. isgreater Is greater
bool isgreater(float x, float y);
bool isgreater(double x, double y);
bool isgreater(long double x, long double y);
bool isgreater(float x, float y);
bool isgreater(double x, double y);
bool isgreater(long double x, long double y);

66. isgreaterequal Is greater or equal
bool isgreaterequal(float x, float y);
bool isgreaterequal(double x, double y);
bool isgreaterequal(long double x, long double y);
bool isgreaterequal(float x, float y);
bool isgreaterequal(double x, double y);
bool isgreaterequal(long double x, long double y);

67. isless Is less
bool isless(float x, float y);
bool isless(double x, double y);
bool isless(long double x, long double y);
bool isless(float x, float y);
bool isless(double x, double y);
bool isless(long double x, long double y);

68. islessequal Is less or equal
bool islessequal(float x, float y);
bool islessequal(double x, double y);
bool islessequal(long double x, long double y);
bool islessequal(float x, float y);
bool islessequal(double x, double y);

bool islessequal(long double x, long double y);
69. islessgreater Is less or greater
bool islessgreater(float x, float y);
bool islessgreater(double x, double y);
bool islessgreater(long double x, long double y);
bool islessgreater(float x, float y);
bool islessgreater(double x, double y);
bool islessgreater(long double x, long double y);
70. isunordered Is unordered
bool isunordered(float x, float y);
bool isunordered(double x, double y);
bool isunordered(long double x, long double y);
bool isunordered(float x, float y);
bool isunordered(double x, double y);
bool isunordered(long double x, long double y);

<numeric>

1. accumulate Accumulate values in range
template <class InputIterator, class T> T accumulate (InputIterator first, InputIterator last, T init);
2. adjacent_difference Compute adjacent difference of range
template <class InputIterator, class OutputIterator> OutputIterator adjacent_difference (InputIterator first, InputIterator last, OutputIterator result);
3. inner_product Compute cumulative inner product of range
template <class InputIterator1, class InputIterator2, class T> T inner_product (InputIterator1 first1, InputIterator1 last1, InputIterator2 first2, T init);
4. partial_sum Compute partial sums of range
template <class InputIterator, class OutputIterator> OutputIterator partial_sum (InputIterator first, InputIterator last, OutputIterator result);
5. iota Store increasing sequence
template <class ForwardIterator, class T> void iota (ForwardIterator first, ForwardIterator last, T val);

<complex>

1. complex Complex number class
template <class T> class complex;

2. real Real part of complex
template<class T> T real(const complex<T>& x);
double real(ArithmeticType x);
template<class T> T real(const complex<T>& x);

3. imag Imaginary part of complex
template<class T> T imag(const complex<T>& x);
double imag(ArithmeticType x);
template<class T> T imag(const complex<T>& x);

4. abs Absolute value of complex
template<class T> T abs (const complex<T>& x);

5. arg Phase angle of complex
template<class T> T arg(const complex<T>& x);
double arg(ArithmeticType x);
template<class T> T arg(const complex<T>& x);

6. norm Norm of complex
template<class T> T norm(const complex<T>& x);
double norm(ArithmeticType x);
template<class T> T norm(const complex<T>& x);

7. conj Complex conjugate
template<class T> complex<T> conj(const complex<T>& x);
complex<double> conj(ArithmeticType x);
template<class T> complex<T> conj(const complex<T>& x);

8. polar Complex from polar components
template<class T> complex<T> polar (const T& rho, const T& theta = 0);

9. proj Complex projection.
template<class T> complex<T> proj(const complex<T>& x);
complex<double> conj(ArithmeticType x);
template<class T> complex<T> proj(const complex<T>& x);

10. cos Cosine of complex
template<class T> complex<T> cos (const complex<T>& x);

11. cosh Hyperbolic cosine of complex
template<class T> complex<T> cosh (const complex<T>& x);

12. exp Exponential of complex
template<class T> complex<T> exp (const complex<T>& x);

13. log Natural logarithm of complex
template<class T> complex<T> log (const complex<T>& x);

14. log10 Common logarithm of complex
template<class T> complex<T> log10 (const complex<T>& x);

15. pow Power of complex
template<class T> complex<T> pow(const complex<T>& x, const complex<T>& y);
template<class T> complex<T> pow(const complex<T>& x, const T& y);
template<class T> complex<T> pow(const T& x, const complex<T>& y);

16. sin Sine of complex
template<class T> complex<T> sin (const complex<T>& x);

17. sinh Hyperbolic sine of complex
template<class T> complex<T> sinh (const complex<T>& x);

18. sqrt Square root of complex
template<class T> complex<T> sqrt (const complex<T>& x);

19. tan Tangent of complex
template<class T> complex<T> tan (const complex<T>& x);

20. tanh Hyperbolic tangent of complex
template<class T> complex<T> tanh (const complex<T>& x);

21. acos Arc cosine of complex
template<class T> complex<T> acos (const complex<T>& x);

22. acosh Arc hyperbolic cosine of complex
template<class T> complex<T> acosh (const complex<T>& x);

23. asin Arc sine of complex
template<class T> complex<T> asin (const complex<T>& x);

24. asinh Arc hyperbolic sine of complex
template<class T> complex<T> asinh (const complex<T>& x);

25. atan Arc tangent of complex
template<class T> complex<T> atan (const complex<T>& x);

26. atanh Arc hyperbolic tangent of complex
template<class T> complex<T> atanh (const complex<T>& x);

27. complex operators Complex number operators
complex& operator= (const T& val); complex& operator+= (const T& val); complex& operator-= (const T& val); complex& operator*= (const T& val); complex& operator/= (const T& val); complex& operator= (const complex& rhs); template<class X> complex& operator= (const complex<X>& rhs);
template<class X> complex& operator+= (const complex<X>& rhs);
template<class X> complex& operator-= (const complex<X>& rhs);
template<class X> complex& operator*= (const complex<X>& rhs);
template<class X> complex& operator/= (const complex<X>& rhs);

<string>

1. basic_string Generic string class
template < class charT, class traits = char_traits<charT>, // basic_string::traits_type class Alloc = allocator<charT> // basic_string::allocator_type > class basic_string;

2. char_traits Character traits
template <class charT> struct char_traits;
template <> struct char_traits<char>;
template <> struct char_traits<wchar_t>;
template <> struct char_traits<char16_t>;
template <> struct char_traits<char32_t>;

3. string String class
typedef basic_string<char> string;

4. u16string String of 16-bit characters
typedef basic_string<char16_t> u16string;

5. u32string String of 32-bit characters

typedef basic_string<char32_t> u32string;

6. wstring Wide string
typedef basic_string<wchar_t> wstring;

7. stoi Convert string to integer
int stoi (const string& str, size_t* idx = 0, int base = 10); int stoi (const wstring& str, size_t* idx = 0, int base = 10);

8. stol Convert string to long int
long stol (const string& str, size_t* idx = 0, int base = 10); long stol (const wstring& str, size_t* idx = 0, int base = 10);

9. stoul Convert string to unsigned integer
unsigned long stoul (const string& str, size_t* idx = 0, int base = 10); unsigned long stoul (const wstring& str, size_t* idx = 0, int base = 10);

10. stoll Convert string to long long
long long stoll (const string& str, size_t* idx = 0, int base = 10); long long stoll (const wstring& str, size_t* idx = 0, int base = 10);

11. stoull Convert string to unsigned long long
unsigned long long stoull (const string& str, size_t* idx = 0, int base = 10); unsigned long long stoull (const wstring& str, size_t* idx = 0, int base = 10);

12. stof Convert string to float
float stof (const string& str, size_t* idx = 0); float stof (const wstring& str, size_t* idx = 0);

13. stod Convert string to double
double stod (const string& str, size_t* idx = 0); double stod (const wstring& str, size_t* idx = 0);

14. stold Convert string to long double
long double stold (const string& str, size_t* idx = 0); long double stold (const wstring& str, size_t* idx = 0);

15. to_string Convert numerical value to string
string to_string (int val); string to_string (long val); string to_string (long long val); string to_string (unsigned val); string to_string (unsigned long val); string to_string (unsigned long long val); string to_string (float val); string to_string (double val); string to_string (long double val);

16. to_wstring Convert numerical value to wide string
wstring to_wstring (int val); wstring to_wstring (long val); wstring to_wstring (long long val); wstring to_wstring (unsigned val); wstring to_wstring (unsigned long val); wstring to_wstring (unsigned long long val); wstring to_wstring (float val); wstring to_wstring (double val); wstring to_wstring (long double val);

17. begin Iterator to beginning
template <class Container> auto begin(Container& cont) -> decltype(cont.begin());
template <class Container> auto begin(const Container& cont) -> decltype(cont.begin());
template <class T, size_t N> T* begin(T(&arr)[N]);
template <class Container> auto begin(Container& cont) -> decltype(cont.begin());
template <class Container> auto begin(const Container& cont) -> decltype(cont.begin());

18. end Iterator to end
template <class Container> auto end(Container& cont) -> decltype(cont.end());
template <class Container> auto end(const Container& cont) -> decltype(cont.end());
template <class T, size_t N> T* end(T(&arr)[N]);
template <class Container> auto end(Container& cont) -> decltype(cont.end());
template <class Container> auto end(const Container& cont) -> decltype(cont.end());

<cstring>

1. memcpy Copy block of memory
void * memcpy (void * destination, const void * source, size_t num);

2. memmove Move block of memory
void * memmove (void * destination, const void * source, size_t num);

3. strcpy Copy string
char * strcpy (char * destination, const char * source);

4. strncpy Copy characters from string
char * strncpy (char * destination, const char * source, size_t num);

5. strcat Concatenate strings
char * strcat (char * destination, const char * source);

6. strncat Append characters from string
char * strncat (char * destination, const char * source, size_t num);

7. memcmp Compare two blocks of memory
int memcmp (const void * ptr1, const void * ptr2, size_t num);

8. strcmp Compare two strings
int strcmp (const char * str1, const char * str2);

9. strcoll Compare two strings using locale
int strcoll (const char * str1, const char * str2);

10. strncmp Compare characters of two strings
int strncmp (const char * str1, const char * str2, size_t num);

11. strxfrm Transform string using locale
size_t strxfrm (char * destination, const char * source, size_t num);

12. memchr Locate character in block of memory
const void * memchr (const void * ptr, int value, size_t num); void * memchr (void * ptr, int value, size_t num);

13. strchr Locate first occurrence of character in string
const char * strchr (const char * str, int character); char * strchr (char * str, int character);

14. strcspn Get span until character in string
size_t strcspn (const char * str1, const char * str2);

15. strpbrk Locate characters in string
const char * strpbrk (const char * str1, const char * str2); char * strpbrk (char * str1, const char * str2);

16. strrchr Locate last occurrence of character in string
const char * strrchr (const char * str, int character); char * strrchr (char * str, int character);

17. strspn Get span of character set in string
size_t strspn (const char * str1, const char * str2);

18. strstr Locate substring
const char * strstr (const char * str1, const char * str2); char * strstr (char * str1, const char * str2);

19. strtok Split string into tokens
char * strtok (char * str, const char * delimiters);

20. memset Fill block of memory
void * memset (void * ptr, int value, size_t num);

21. strerror Get pointer to error message string
char * strerror (int errnum);

22. strlen Get string length
size_t strlen (const char * str);

23. NULL Null pointer
23. size_t Unsigned integral type

<cctype>

1. isalnum Check if character is alphanumeric
int isalnum (int c);

2. isalpha Check if character is alphabetic
int isalpha (int c);

3. isblank Check if character is blank
int isblank (int c);

4. iscntrl Check if character is a control character
int iscntrl (int c);

5. isdigit Check if character is decimal digit
int isdigit (int c);

6. isgraph Check if character has graphical representation
int isgraph (int c);

7. islower Check if character is lowercase letter
int islower (int c);

8. isprint Check if character is printable
int isprint (int c);

9. ispunct Check if character is a punctuation character
int ispunct (int c);

10. isspace Check if character is a white-space
int isspace (int c);

11. isupper Check if character is uppercase letter
int isupper (int c);

12. isxdigit Check if character is hexadecimal digit
int isxdigit (int c);

13. tolower Convert uppercase letter to lowercase
int tolower (int c);

14. toupper Convert lowercase letter to uppercase
int toupper (int c);

<ios>

1. basic_ios Base class for streams (type-dependent components)
template <class charT, class traits = char_traits<charT> > class basic_ios;
2. fpos Stream position class template
template <class stateT> class fpos;
3. ios Base class for streams (type-dependent components)
typedef basic_ios<char> ios;
4. ios_base Base class for streams
class ios_base;
5. wios Base class for wide character streams
typedef basic_ios<wchar_t> ios;
6. io_errc Input/output error conditions
enum class io_errc;
7. streamoff Stream offset type
7. streampos Stream position type
typedef fpos<mbstate_t> streampos;
8. streamsize Stream size type
8. wstreampos Wide stream position type
typedef fpos<mbstate_t> wstreampos;
9. boolalpha Alphanumerical bool values
ios_base& boolalpha (ios_base& str);
10. showbase Show numerical base prefixes
ios_base& showbase (ios_base& str);
11. showpoint Show decimal point
ios_base& showpoint (ios_base& str);
12. showpos Show positive signs
ios_base& showpos (ios_base& str);
13. skipws Skip whitespaces
ios_base& skipws (ios_base& str);
14. unitbuf Flush buffer after insertions
ios_base& unitbuf (ios_base& str);
15. uppercase Generate upper-case letters
ios_base& uppercase (ios_base& str);
16. noboolalpha No alphanumerical bool values
ios_base& noboolalpha (ios_base& str);
17. noshowbase Do not show numerical base prefixes
ios_base& noshowbase (ios_base& str);
18. noshowpoint Do not show decimal point
ios_base& noshowpoint (ios_base& str);
19. noshowpos Do not show positive signs
ios_base& noshowpos (ios_base& str);
20. noskipws Do not skip whitespaces
ios_base& noskipws (ios_base& str);
21. nounitbuf Do not force flushes after insertions
ios_base& nounitbuf (ios_base& str);
22. nouppercase Do not generate upper case letters
ios_base& nouppercase (ios_base& str);
23. dec Use decimal base
ios_base& dec (ios_base& str);
24. hex Use hexadecimal base
ios_base& hex (ios_base& str);
25. oct Use octal base
ios_base& oct (ios_base& str);
26. fixed Use fixed floating-point notation
ios_base& fixed (ios_base& str);
27. scientific Use scientific floating-point notation
ios_base& scientific (ios_base& str);
28. internal Adjust field by inserting characters at an internal position
ios_base& internal (ios_base& str);
29. left Adjust output to the left
ios_base& left (ios_base& str);
30. right Adjust output to the right
ios_base& right (ios_base& str);
31. iostream_category Return iostream category
const error_category& iostream_category();

<iomanip>

1. setiosflags Set format flags
setiosflags (ios_base::fmtflags mask);

2. resetiosflags Reset format flags
resetiosflags (ios_base::fmtflags mask);

3. setbase Set basefield flag
setbase (int base);

4. setfill Set fill character
setfill (char_type c);

5. setprecision Set decimal precision
setprecision (int n);

6. setw Set field width
setw (int n);

7. get_money Get monetary value
template <class moneyT> get_money (moneyT& mon, bool intl = false);

8. put_money Put monetary value
template <class moneyT> put_money (const moneyT& mon, bool intl = false);

9. get_time Get date and time
template <class charT> get_time (struct tm* tmb, const charT* fmt);

10. put_time Put date and time
template <class charT> put_time (const struct tm* tmb, const charT* fmt);

<cstdlib>

1. atof Convert string to double
double atof (const char* str);

2. atoi Convert string to integer
int atoi (const char * str);

3. atol Convert string to long integer
long int atol (const char * str);

4. atoll Convert string to long long integer
long long int atoll (const char * str);

5. strtod Convert string to double
double strtod (const char* str, char** endptr);

6. strtof Convert string to float
float strtof (const char* str, char** endptr);

7. strtol Convert string to long integer
long int strtol (const char* str, char** endptr, int base);

8. strtold Convert string to long double
long double strtold (const char* str, char** endptr);

9. strtoll Convert string to long long integer
long long int strtoll (const char* str, char** endptr, int base);

10. strtoul Convert string to unsigned long integer
unsigned long int strtoul (const char* str, char** endptr, int base);

11. strtoull Convert string to unsigned long long integer
unsigned long long int strtoull (const char* str, char** endptr, int base);

12. rand Generate random number
int rand (void);

13. srand Initialize random number generator
void srand (unsigned int seed);

14. calloc Allocate and zero-initialize array
void* calloc (size_t num, size_t size);

15. free Deallocate memory block
void free (void* ptr);

16. malloc Allocate memory block
void* malloc (size_t size);

17. realloc Reallocate memory block
void* realloc (void* ptr, size_t size);

18. abort Abort current process
void abort() noexcept;

19. atexit Set function to be executed on exit
int atexit(void(*func)(void)) noexcept;

20. at_quick_exit Set function to be executed on quick
int at_quick_exit(void(*func)(void)) noexcept;

21. exit Terminate calling process
void exit(int status);

22. getenv Get environment string
char* getenv (const char* name);

23. quick_exit Terminate calling process quick
void quick_exit(int status) noexcept;

24. system Execute system command
int system (const char* command);

25. _Exit Terminate calling process
void _Exit(int status) noexcept;

26. bsearch Binary search in array
void* bsearch (const void* key, const void* base, size_t num, size_t size, int (*compar)(const void*,const void*));

27. qsort Sort elements of array
void qsort (void* base, size_t num, size_t size, int (*compar)(const void*,const void*));

28. abs Absolute value
int abs(int n);
long int abs(long int n);
long long int abs(long long int n);

29. div Integral division
div_t div(int numer, int denom);
ldiv_t div(long int numer, long int denom);
lldiv_t div(long long int numer, long long int denom);

30. labs Absolute value
long int labs (long int n);

31. ldiv Integral division
ldiv_t ldiv (long int numer, long int denom);

32. llabs Absolute value
long long int llabs (long long int n);

33. lldiv Integral division
lldiv_t lldiv (long long int numer, long long int denom);

34. mblen Get length of multibyte character
int mblen (const char* pmb, size_t max);

35. mbtowc Convert multibyte sequence to wide character
int mbtowc (wchar_t* pwc, const char* pmb, size_t max);

36. wctomb Convert wide character to multibyte sequence
int wctomb (char* pmb, wchar_t wc);

37. mbstowcs Convert multibyte string to wide-character string
size_t mbstowcs (wchar_t* dest, const char* src, size_t max);

38. wcstombs Convert wide-character string to multibyte string
size_t wcstombs (char* dest, const wchar_t* src, size_t max);

<sstream>

1. basic_istringstream Input string stream
template < class charT, // basic_istringstream::char_type class traits = char_traits<charT>, // basic_istringstream::traits_type class Alloc = allocator<charT> // basic_istringstream::allocator_type > class basic_istringstream;

2. basic_ostringstream Output string stream
template < class charT, // basic_ostringstream::char_type class traits = char_traits<charT>, // basic_ostringstream::traits_type class Alloc = allocator<charT> // basic_ostringstream::allocator_type > class basic_ostringstream;

3. basic_stringstream String stream
template < class charT, // basic_stringstream::char_type class traits = char_traits<charT>, // basic_stringstream::traits_type class Alloc = allocator<charT> // basic_stringstream::allocator_type > class basic_stringstream;

4. basic_stringbuf String stream buffer
template < class charT, // basic_stringbuf::char_type class traits = char_traits<charT>, // basic_stringbuf::traits_type class Alloc = allocator<charT> // basic_stringbuf::allocator_type > class basic_stringbuf;

5. istringstream Input string stream
typedef basic_istringstream<char> istringstream;

6. ostringstream Output string stream
typedef basic_ostringstream<char> ostringstream;

7. stringstream Input/output string stream
typedef basic_stringstream<char> stringstream;

8. stringbuf String stream buffer
typedef basic_stringbuf<char> stringbuf;

9. wistringstream Input string stream (wide)
typedef basic_istringstream<wchar_t> wistringstream;

10. wostringstream Output string stream (wide)
typedef basic_ostringstream<wchar_t> wostringstream;

11. wstringstream Input/output string stream (wide)
typedef basic_stringstream<wchar_t> wstringstream;

12. wstringbuf String stream buffer (wide)
typedef basic_stringbuf<wchar_t> wstringbuf;

<climits>

name	expresses	value*
CHAR_BIT	Number of bits in a char object (byte)	8 or greater*
SCHAR_MIN	Minimum value for an object of type signed char	-127 (-2^7+1) or less*
SCHAR_MAX	Maximum value for an object of type signed char	127 (2^7-1) or greater*
UCHAR_MAX	Maximum value for an object of type unsigned char	255 (2^8-1) or greater*
CHAR_MIN	Minimum value for an object of type char	either SCHAR_MIN or 0
CHAR_MAX	Maximum value for an object of type char	either SCHAR_MAX or UCHAR_MAX
MB_LEN_MAX	Maximum number of bytes in a multibyte character, for any locale	1 or greater*
SHRT_MIN	Minimum value for an object of type short int	-32767 ($-2^{15}+1$) or less*
SHRT_MAX	Maximum value for an object of type short int	32767 ($2^{15}-1$) or greater*
USHRT_MAX	Maximum value for an object of type unsigned short int	65535 ($2^{16}-1$) or greater*
INT_MIN	Minimum value for an object of type int	-32767 ($-2^{15}+1$) or less*
INT_MAX	Maximum value for an object of type int	32767 ($2^{15}-1$) or greater*
UINT_MAX	Maximum value for an object of type unsigned int	65535 ($2^{16}-1$) or greater*
LONG_MIN	Minimum value for an object of type long int	-2147483647 ($-2^{31}+1$) or less*
LONG_MAX	Maximum value for an object of type long int	2147483647 ($2^{31}-1$) or greater*
ULONG_MAX	Maximum value for an object of type unsigned long int	4294967295 ($2^{32}-1$) or greater*
LLONG_MIN	Minimum value for an object of type long long int	-9223372036854775807 ($-2^{63}+1$) or less*
LLONG_MAX	Maximum value for an object of type long long int	9223372036854775807 ($2^{63}-1$) or greater*
ULLONG_MAX	Maximum value for an object of type unsigned long long int	18446744073709551615 ($2^{64}-1$) or greater*

<cfloats>

name	value	stands for	expresses
FLT_RADIX	2 or greater	RADIX	Base for all floating-point types (float, double and long double).
FLT_MANT_DIG DBL_MANT_DIG LDBL_MANT_DIG		MANTissa DIGits	Precision of *significand*, i.e. the number of digits that conform the *significand*.
FLT_DIG DBL_DIG LDBL_DIG	6 or greater 10 or greater 10 or greater	DIGits	Number of **decimal digits** that can be rounded into a floating-point and back without change in the number of decimal digits.
FLT_MIN_EXP DBL_MIN_EXP LDBL_MIN_EXP		MINimum EXPonent	Minimum negative integer value for the *exponent* that generates a normalized floating-point number.
FLT_MIN_10_EXP DBL_MIN_10_EXP LDBL_MIN_10_EXP	-37 or smaller -37 or smaller -37 or smaller	MINimum base-10 EXPonent	Minimum negative integer value for the *exponent* of a base-10 expression that would generate a normalized floating-point number.
FLT_MAX_EXP DBL_MAX_EXP LDBL_MAX_EXP		MAXimum EXPonent	Maximum integer value for the *exponent* that generates a normalized floating-point number.
FLT_MAX_10_EXP DBL_MAX_10_EXP LDBL_MAX_10_EXP	37 or greater 37 or greater 37 or greater	MAXimum base-10 EXPonent	Maximum integer value for the *exponent* of a base-10 expression that would generate a normalized floating-point number.
FLT_MAX DBL_MAX LDBL_MAX	1E+37 or greater 1E+37 or greater 1E+37 or greater	MAXimum	Maximum finite representable floating-point number.
FLT_EPSILON DBL_EPSILON LDBL_EPSILON	1E-5 or smaller 1E-9 or smaller 1E-9 or smaller	EPSILON	Difference between 1 and the least value greater than 1 that is representable.
FLT_MIN DBL_MIN LDBL_MIN	1E-37 or smaller 1E-37 or smaller 1E-37 or smaller	MINimum	Minimum representable floating-point number.
FLT_ROUNDS		ROUND	Rounding behavior. Possible values: -1 undetermined 0 toward zero 1 to nearest 2 toward positive infinity 3 toward negative infinity Applies to all floating-point types (float, double and long double).
FLT_EVAL_METHOD		EVALuation METHOD	Properties of the evaluation format. Possible values: -1 undetermined 0 evaluate just to the range and precision of the type 1 evaluate float and double as double, and long double as long double. 2 evaluate all as long double Other negative values indicate an implementation-defined behavior. Applies to all floating-point types (float, double and long double).
DECIMAL_DIG		DECIMAL DIGits	Number of decimal digits that can be rounded into a floating-point type and back again to the same decimal digits, without loss in precision.

<ASCII table>

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□