

Compiler Package

Application Note: <STL V.1.00.00>

R20UT0075EJ0100

Rev.1.00

Jul. 30, 2010

Lists of Supported Functions

This document explains the usage of STL for the Renesas C/C++ compiler (hereafter referred to as 'STL') V.1.00.00.

Contents

1.	List of Headers for the STL	2
2.	<algorithm>	3
3.	<bitset>.....	9
4.	<complex>.....	11
5.	<deque>	17
6.	<exception>.....	19
7.	<functional>.....	20
8.	<iterator>	22
9.	<limits>	24
10.	<list>.....	25
11.	<map>	27
12.	<memory>	31
13.	<numeric>	32
14.	<queue>	33
15.	<set>	34
16.	<stack>.....	38
17.	<stdexcept>.....	38
18.	<string>.....	39
19.	<typeinfo>	42
20.	<utility>.....	42
21.	<valarray>	43
22.	<vector>	48

1. List of Headers for the STL

This document lists the functions supported by STL V.1.00.00 to the following C/C++ compiler manufactured by Renesas Electronics Corporation (hereafter referred to as 'Renesas Electronics').

- RX family C/C++ compiler V.1.00.00 or a later version
- SuperH family C/C++ compiler V.9.04.00 or a later version

This STL supports the following headers.

No.	Header Name	Methods	Remarks
1	<algorithm>	Yes	–
2	<bitset>	Yes	–
3	<complex>	Yes	–
4	<deque>	Yes	–
5	<exception>	Yes	–
6	<functional>	Yes	–
7	<iterator>	Yes	–
8	<limits>	Yes	–
9	<list>	Yes	–
10	<map>	Yes	–
11	<memory>	Yes	–
12	<numeric>	Yes	–
13	<queue>	Yes	–
14	<set>	Yes	–
15	<stack>	Yes	–
16	<stdexcept>	Yes	–
17	<string>	Yes	Although <string> is normally used, <string> is used here to avoid the conflict with the name in EC++.
	<string>	Yes	
18	<typeinfo>	Yes	–
19	<utility>	Yes	–
20	<valarray>	Yes	–
21	<vector>	Yes	–
22	<cassert>	No	Refer to <assert.h>
23	<cctype>	No	Refer to <ctype.h>
24	<cerrno>	No	Refer to <errno.h>
25	<cfloat>	No	Refer to <float.h>
26	<climits>	No	Refer to <limits.h>
27	<cmath>	No	Refer to <math.h>
28	<csetjmp>	No	Refer to <setjmp.h>
29	<cstdarg>	No	Refer to <stdarg.h>
30	<cstddef>	No	Refer to <stddef.h>
31	<cstdio>	No	Refer to <stdio.h>
32	<cstdlib>	No	Refer to <stdlib.h>
33	<cstring>	No	Refer to <string.h>
34	<new.h>	No	Refer to <new>

Next, the methods supported by the individual headers are listed.

2. <algorithm>

The methods supported by <algorithm> are listed below.

List of Methods for the Class algorithm

1	template <class _Tp> void swap(_Tp& __a, _Tp& __b)
2	void iter_swap(_ForwardIter1 __i1, _ForwardIter2 __i2)
3	template <class _Tp> inline const _Tp& (min)(const _Tp& __a, const _Tp& __b)
4	template <class _Tp> inline const _Tp& (min)(const _Tp& __a, const _Tp& __b, _Compare __comp)
5	template <class _Tp> inline const _Tp& (max)(const _Tp& __a, const _Tp& __b)
6	template <class _Tp> inline const _Tp& (max)(const _Tp& __a, const _Tp& __b, _Compare __comp)
7	_OutputIter copy(_InputIter __first, _InputIter __last, _OutputIter __result)
8	_OutputIter copy_backward(_InputIter __first, _InputIter __last, _OutputIter __result)
9	template <class _ForwardIter, class _Tp> void fill(_ForwardIter __first, _ForwardIter __last, const _Tp& __val)
10	void fill(unsigned char* __first, unsigned char* __last, const unsigned char& __val)
11	void fill(signed char* __first, signed char* __last, const signed char& __val)
12	void fill(char* __first, char* __last, const char& __val)
13	template <class _OutputIter, class _Size, class _Tp> void fill_n(_OutputIter __first, _Size __n, const _Tp& __val)
14	template <class _InputIter1, class _InputIter2> ::stlpmx_std::pair<_InputIter1, _InputIter2> mismatch(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2)
15	template <class _InputIter1, class _InputIter2, class _BinaryPredicate> ::stlpmx_std::pair<_InputIter1, _InputIter2> mismatch(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _BinaryPredicate __binary_pred)
16	template <class _InputIter1, class _InputIter2> bool equal(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2)
17	template <class _InputIter1, class _InputIter2, class _BinaryPredicate> bool equal(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _BinaryPredicate __binary_pred)
18	template <class _InputIter1, class _InputIter2> bool lexicographical_compare(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2);
19	template <class _InputIter1, class _InputIter2, class _Compare> bool lexicographical_compare(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _Compare __comp);
20	bool lexicographical_compare(const unsigned char* __first1, const unsigned char* __last1, const unsigned char* __first2, const unsigned char* __last2)

21	int lexicographical_compare_3way(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2);
22	template <class _InputIter, class _Tp> iterator_traits<_InputIter> ::difference_type count(_InputIter __first, _InputIter __last, const _Tp& __val)
23	template <class _InputIter, class _Tp> _InputIter find(_InputIter __first, _InputIter __last, const _Tp& __val)
24	_InputIter find_if(_InputIter __first, _InputIter __last, _Predicate __pred)
25	_ForwardIter1 search(_ForwardIter1 __first1, _ForwardIter1 __last1, _ForwardIter2 __first2, _ForwardIter2 __last2, _BinaryPred __predicate);
26	template <class _ForwardIter1, class _ForwardIter2, class _BinaryPredicate> _ForwardIter1 find_end(_ForwardIter1 __first1, _ForwardIter1 __last1, _ForwardIter2 __first2, _ForwardIter2 __last2, _BinaryPredicate __comp);
27	template <class _ForwardIter, class _Tp> void replace(_ForwardIter __first, _ForwardIter __last, const _Tp& __old_value, const _Tp& __new_value)
28	template <class _InputIter, class _Function> nline _Function for_each(_InputIter __first, _InputIter __last, _Function __f)
29	template <class _ForwardIter, class _BinaryPredicate> _ForwardIter adjacent_find(_ForwardIter __first, _ForwardIter __last, _BinaryPredicate __binary_pred)
30	template <class _ForwardIter> _ForwardIter adjacent_findadjacent_find(_ForwardIter __first, _ForwardIter __last)
31	template <class _InputIter, class _Tp, class _Size> _STLP_INLINE_LOOP void count(_InputIter __first, _InputIter __last, const _Tp& __val, _Size& __n)
32	template <class _InputIter, class _Predicate> _STLP_INLINE_LOOP _STLP_DIFFERENCE_TYPE(_InputIter) count_if(_InputIter __first, _InputIter __last, _Predicate __pred)
33	template <class _InputIter, class _Predicate, class _Size> _STLP_INLINE_LOOP void count_if(_InputIter __first, _InputIter __last, _Predicate __pred, _Size& __n)
34	template <class _ForwardIter1, class _ForwardIter2> _ForwardIter1 search(_ForwardIter1 __first1, _ForwardIter1 __last1, _ForwardIter2 __first2, _ForwardIter2 __last2);
35	template <class _ForwardIter, class _Integer, class _Tp> _ForwardIter search_n(_ForwardIter __first, _ForwardIter __last, _Integer __count, const _Tp& __val);
36	template <class _ForwardIter, class _Integer, class _Tp, class _BinaryPred> _ForwardIter search_n(_ForwardIter __first, _ForwardIter __last, _Integer __count, const _Tp& __val, _BinaryPred __binary_pred);
37	template <class _InputIter, class _ForwardIter> inline _InputIter find_first_of(_InputIter __first1, _InputIter __last1, _ForwardIter __first2, _ForwardIter __last2)
38	template <class _InputIter, class _ForwardIter, class _BinaryPredicate> find_first_of(_InputIter __first1, _InputIter __last1,
39	template <class _ForwardIter1, class _ForwardIter2> _ForwardIter1 find_end(_ForwardIter1 __first1, _ForwardIter1 __last1, _ForwardIter2 __first2, _ForwardIter2 __last2);
40	template <class _ForwardIter1, class _ForwardIter2> _STLP_INLINE_LOOP _ForwardIter2 swap_ranges(_ForwardIter1 __first1, _ForwardIter1 __last1, _ForwardIter2 __first2)

41	template <class _InputIter, class _OutputIter, class _UnaryOperation> _OutputIter transform(_InputIter __first, _InputIter __last, _OutputIter __result, _UnaryOperation __opr)
42	template <class _InputIter1, class _InputIter2, class _OutputIter, class _BinaryOperation> _OutputIter transform(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _OutputIter __result, _BinaryOperation __binary_op)
43	template <class _ForwardIter, class _Predicate, class _Tp> _ForwardIter remove_if(_ForwardIter __first, _ForwardIter __last, _Predicate __pred)
44	template <class _InputIter, class _OutputIter, class _Tp> _OutputIter replace_copy(_InputIter __first, _InputIter __last, _OutputIter __result, const _Tp& __old_value, const _Tp& __new_value)
45	template <class _Iterator, class _OutputIter, class _Predicate, class _Tp> _OutputIter replace_copy_if(_Iterator __first, _Iterator __last, _OutputIter __result, _Predicate __pred, const _Tp& __new_value)
46	template <class _ForwardIter, class _Generator> void generate(_ForwardIter __first, _ForwardIter __last, _Generator __gen)
47	template <class _OutputIter, class _Size, class _Generator> void generate_n(_OutputIter __first, _Size __n, _Generator __gen)
48	template <class _InputIter, class _OutputIter, class _Tp> _OutputIter remove_copy(_InputIter __first, _InputIter __last, _OutputIter __result, const _Tp& __val)
49	template <class _InputIter, class _OutputIter, class _Predicate> _OutputIter remove_copy_if(_InputIter __first, _InputIter __last, _OutputIter __result, _Predicate __pred)
50	template <class _ForwardIter, class _Tp> _ForwardIter remove(_ForwardIter __first, _ForwardIter __last, const _Tp& __val)
51	template <class _ForwardIter, class _Predicate> void replace_if(_ForwardIter __first, _ForwardIter __last, _Predicate __pred, const _Tp& __new_value)
52	template <class _InputIter, class _OutputIter> _OutputIter unique_copy(_InputIter __first, _InputIter __last, _OutputIter __result)
53	template <class _InputIter, class _OutputIter, class _BinaryPredicate> _OutputIter unique_copy(_InputIter __first, _InputIter __last, _OutputIter __result, _BinaryPredicate __binary_pred)
54	template <class _ForwardIter, class _BinaryPredicate> _ForwardIter unique(_ForwardIter __first, _ForwardIter __last, _BinaryPredicate __binary_pred)
55	template <class _BidirectionalIter> void reverse(_BidirectionalIter __first, _BidirectionalIter __last)
56	template <class _BidirectionalIter, class _OutputIter> _OutputIter reverse_copy(_BidirectionalIter __first, _BidirectionalIter __last, _OutputIter __result)
57	template <class _ForwardIter> void rotate(_ForwardIter __first, _ForwardIter __middle, _ForwardIter __last)
58	template <class _ForwardIter, class _OutputIter> _OutputIter rotate_copy(_ForwardIter __first, _ForwardIter __middle, _ForwardIter __last, _OutputIter __result)
59	template <class _RandomAccessIter> void random_shuffle(_RandomAccessIter __first, _RandomAccessIter __last)
60	template <class _RandomAccessIter, class _RandomNumberGenerator> void random_shuffle(_RandomAccessIter __first, _RandomAccessIter __last, _RandomNumberGenerator& __rand)

61	template <class _ForwardIter, class _OutputIter, class _Distance> OutputIter random_sample_n(_ForwardIter __first, _ForwardIter __last, _OutputIter __out_ite, const _Distance __n)
62	template <class _ForwardIter, class _OutputIter, class _Distance, class _RandomNumberGenerator> _OutputIter random_sample_n(_ForwardIter __first, _ForwardIter __last, _OutputIter __out_ite, const _Distance __n, _RandomNumberGenerator& __rand)
63	template <class _InputIter, class _RandomAccessIter> _RandomAccessIter random_sample(_InputIter __first, _InputIter __last, _RandomAccessIter __out_first, _RandomAccessIter __out_last)
64	template <class _InputIter, class _RandomAccessIter, class _RandomNumberGenerator> _RandomAccessIter random_sample(_InputIter __first, _InputIter __last, _RandomAccessIter __out_first, _RandomAccessIter __out_last, _RandomNumberGenerator& __rand)
65	template <class _ForwardIter, class _Predicate> _ForwardIter partition(_ForwardIter __first, _ForwardIter __last, _Predicate __pred)
66	template <class _ForwardIter, class _Predicate> _ForwardIter stable_partition(_ForwardIter __first, _ForwardIter __last, _Predicate __pred)
67	template <class _RandomAccessIter> void sort(_RandomAccessIter __first, _RandomAccessIter __last)
68	template <class _RandomAccessIter, class _Compare> void sort(_RandomAccessIter __first, _RandomAccessIter __last, _Compare __comp)
69	template <class _RandomAccessIter> void stable_sort(_RandomAccessIter __first, _RandomAccessIter __last)
70	template <class _RandomAccessIter, class _Compare> void stable_sort(_RandomAccessIter __first, _RandomAccessIter __last, _Compare __comp)
71	template <class _RandomAccessIter> void partial_sort(_RandomAccessIter __first, _RandomAccessIter __middle, _RandomAccessIter __last)
72	template <class _RandomAccessIter, class _Compare> void partial_sort(_RandomAccessIter __first, _RandomAccessIter __middle, _RandomAccessIter __last, _Compare __comp)
73	template <class _InputIter, class _RandomAccessIter> _RandomAccessIter partial_sort_copy(_InputIter __first, _InputIter __last, _RandomAccessIter __result_first, _RandomAccessIter __result_last)
74	template <class _InputIter, class _RandomAccessIter, class _Compare> partial_sort_copy(_InputIter __first, _InputIter __last, _RandomAccessIter __result_first, _RandomAccessIter __result_last, _Compare __comp)
75	template <class _RandomAccessIter> void nth_element(_RandomAccessIter __first, _RandomAccessIter __nth, _RandomAccessIter __last)
76	template <class _RandomAccessIter, class _Compare> void nth_element(_RandomAccessIter __first, _RandomAccessIter __nth, _RandomAccessIter __last, _Compare __comp)
77	template <class _ForwardIter, class _Tp> _ForwardIter lower_bound(_ForwardIter __first, _ForwardIter __last, const _Tp& __val)
78	template <class _ForwardIter, class _Tp, class _Compare> _ForwardIter lower_bound(_ForwardIter __first, _ForwardIter __last, const _Tp& __val, _Compare __comp)
79	template <class _ForwardIter, class _Tp> inline _ForwardIter upper_bound(_ForwardIter __first, _ForwardIter __last, const _Tp& __val)
80	template <class _ForwardIter, class _Tp, class _Compare> inline _ForwardIter upper_bound(_ForwardIter __first, _ForwardIter __last, const _Tp& __val, _Compare __comp)

81	<pre>template <class _ForwardIter, class _Tp> inline pair<_ForwardIter, _ForwardIter> equal_range(_ForwardIter __first, _ForwardIter __last, const _Tp& __val)</pre>
82	<pre>template <class _ForwardIter, class _Tp, class _Compare> inline pair<_ForwardIter, _ForwardIter> equal_range(_ForwardIter __first, _ForwardIter __last, const _Tp& __val, _Compare __comp)</pre>
83	<pre>template <class _ForwardIter, class _Tp> bool binary_search(_ForwardIter __first, _ForwardIter __last, const _Tp& __val)</pre>
84	<pre>template <class _ForwardIter, class _Tp, class _Compare> bool binary_search(_ForwardIter __first, _ForwardIter __last, const _Tp& __val, _Compare __comp)</pre>
85	<pre>template <class _InputIter1, class _InputIter2, class _OutputIter> _OutputIter merge(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result)</pre>
86	<pre>template <class _InputIter1, class _InputIter2, class _OutputIter, class _Compare> _OutputIter merge(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result, _Compare __comp)</pre>
87	<pre>template <class _BidirectionalIter> void inplace_merge(_BidirectionalIter __first, _BidirectionalIter __middle, _BidirectionalIter __last);</pre>
88	<pre>template <class _BidirectionalIter, class _Compare> void inplace_merge(_BidirectionalIter __first, _BidirectionalIter __middle, _BidirectionalIter __last, _Compare __comp)</pre>
89	<pre>template <class _InputIter1, class _InputIter2> bool includes(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2)</pre>
90	<pre>template <class _InputIter1, class _InputIter2, class _Compare> bool includes(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _Compare __comp)</pre>
91	<pre>template <class _InputIter1, class _InputIter2, class _OutputIter> _OutputIter set_union(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result)</pre>
92	<pre>template <class _InputIter1, class _InputIter2, class _OutputIter, class _Compare> _OutputIter set_union(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result, _Compare __comp)</pre>
93	<pre>template <class _InputIter1, class _InputIter2, class _OutputIter> _OutputIter set_intersection(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result)</pre>
94	<pre>template <class _InputIter1, class _InputIter2, class _OutputIter, class _Compare> _OutputIter set_intersection(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result, _Compare __comp)</pre>
95	<pre>template <class _InputIter1, class _InputIter2, class _OutputIter> _OutputIter set_difference(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result)</pre>
96	<pre>template <class _InputIter1, class _InputIter2, class _OutputIter, class _Compare> _OutputIter set_difference(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result, _Compare __comp)</pre>
97	<pre>template <class _InputIter1, class _InputIter2, class _OutputIter, class _Compare> _OutputIter set_symmetric_difference(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result)</pre>
98	<pre>template <class _InputIter1, class _InputIter2, class _OutputIter> _OutputIter set_symmetric_difference(_InputIter1 __first1, _InputIter1 __last1, _InputIter2 __first2, _InputIter2 __last2, _OutputIter __result, _Compare __comp)</pre>

```
99     template <class _ForwardIter>
      _ForwardIter max_element(_ForwardIter __first, _ForwardIter __last);
100     template <class _ForwardIter, class _Compare>
      _ForwardIter max_element(_ForwardIter __first, _ForwardIter __last, _Compare __comp);
101     template <class _ForwardIter>
      _ForwardIter min_element(_ForwardIter __first, _ForwardIter __last);
102     template <class _ForwardIter, class _Compare>
      _ForwardIter min_element(_ForwardIter __first, _ForwardIter __last, _Compare __comp);
103     template <class _BidirectionalIter>
      bool next_permutation(_BidirectionalIter __first, _BidirectionalIter __last);
104     template <class _BidirectionalIter, class _Compare>
      bool next_permutation(_BidirectionalIter __first, _BidirectionalIter __last, _Compare __comp);
105     template <class _BidirectionalIter>
      bool prev_permutation(_BidirectionalIter __first, _BidirectionalIter __last);
106     template <class _BidirectionalIter, class _Compare>
      bool prev_permutation(_BidirectionalIter __first, _BidirectionalIter __last, _Compare __comp);
107     template <class _RandomAccessIter>
      bool is_heap(_RandomAccessIter __first, _RandomAccessIter __last)
108     template <class _RandomAccessIter, class _StrictWeakOrdering>
      bool is_heap(_RandomAccessIter __first, _RandomAccessIter __last, _StrictWeakOrdering
      __comp)
109     bool is_sorted(_ForwardIter __first, _ForwardIter __last)
110     bool is_sorted(_ForwardIter __first, _ForwardIter __last, _StrictWeakOrdering __comp)
```

3. <bitset>

The methods supported by <bitset> are listed below.

List of Methods for Class bitset

1	reference& operator=(bool __x)
2	reference& operator=(const reference& __j)
3	bool operator~()
4	operator bool()
5	reference& flip()
6	bitset()
7	bitset(unsigned long __val)
8	bitset(const basic_string<_CharT, _Traits, _Alloc>& __s, size_t __pos = 0)
9	bitset(const basic_string<_CharT, _Traits, _Alloc>& __s, size_t __pos, size_t __n)
10	bitset<_Nb>& operator&=(const bitset<_Nb>& __rhs)
11	bitset<_Nb>& operator =(const bitset<_Nb>& __rhs)
12	bitset<_Nb>& operator^=(const bitset<_Nb>& __rhs)
13	bitset<_Nb>& operator<<=(size_t __pos)
14	bitset<_Nb>& operator>>=(size_t __pos)
15	bitset<_Nb>& _Unchecked_set(size_t __pos)
16	bitset<_Nb>& _Unchecked_set(size_t __pos, int __val)
17	bitset<_Nb>& _Unchecked_reset(size_t __pos)
18	bitset<_Nb>& _Unchecked_flip(size_t __pos)
19	bool _Unchecked_test(size_t __pos)
20	bitset<_Nb>& set()
21	bitset<_Nb>& set(size_t __pos)
22	bitset<_Nb>& set(size_t __pos, int __val)
23	bitset<_Nb>& reset()
24	bitset<_Nb>& reset(size_t __pos)
25	bitset<_Nb>& flip()
26	bitset<_Nb>& flip(size_t __pos)
27	bitset<_Nb> operator~()
28	reference operator[](size_t __pos)
29	bool operator[](size_t __pos)
30	unsigned long to_ulong()
31	size_t count()
32	size_t size()
33	bool operator==(const bitset<_Nb>& __rhs)
34	bool operator!=(const bitset<_Nb>& __rhs)
35	bool test(size_t __pos)
36	bool any()
37	bool none()
38	bitset<_Nb> operator<<(size_t __pos)
39	bitset<_Nb> operator>>(size_t __pos)
40	size_t _Find_first() const
41	size_t _Find_next(size_t __prev) const
42	void _M_copy_from_string(const basic_string<_CharT, _Traits, _Alloc>& __s, size_t __pos, size_t __n)
43	void _M_copy_to_string(basic_string<_CharT, _Traits, _Alloc>& __s) const

44	bitset<_Nb> operator&(const bitset<_Nb>& __x, const bitset<_Nb>& __y)
45	bitset<_Nb> operator (const bitset<_Nb>& __x, const bitset<_Nb>& __y)
46	bitset<_Nb> operator^(const bitset<_Nb>& __x, const bitset<_Nb>& __y)

4. <complex>

The methods supported by <complex> are listed below.

List of Methods for Class complex

1	complex()
2	complex(const value_type& __x)
3	complex(const value_type& __x, const value_type& __y)
4	complex(const _Self& __z)
5	template <class _Tp2> explicit complex(const complex<_Tp2>& __z)
6	_Self& operator=(const _Self& __z)
7	template <class _Tp2> _Self& operator=(const complex<_Tp2>& __z)
8	value_type real()
9	value_type imag()
10	_Self& operator= (const value_type& __x)
11	_Self& operator+= (const value_type& __x)
12	_Self& operator-= (const value_type& __x)
13	_Self& operator*= (const value_type& __x)
14	_Self& operator/= (const value_type& __x)
15	static void _STLP_CALL _div(const value_type& __z1_r, const value_type& __z1_i, const value_type& __z2_r, const value_type& __z2_i, value_type& __res_r, value_type& __res_i);
16	static void _STLP_CALL _div(const value_type& __z1_r, const value_type& __z2_r, const value_type& __z2_i, vaue_type& __res_r, es_i);
17	template <class _Tp2> _Self& operator+= (const complex<_Tp2>& __z)
18	template <class _Tp2> _Self& operator-= (const complex<_Tp2>& __z)
19	template <class _Tp2> _Self& operator*= (const complex<_Tp2>& __z)
20	template <class _Tp2> _Self& operator/= (const complex<_Tp2>& __z)
21	_Self& operator+= (const _Self& __z)
22	_Self& operator-= (const _Self& __z)
23	_Self& operator*= (const _Self& __z)
24	_Self& operator/= (const _Self& __z)
25	complex(value_type __x = 0.0f, value_type __y = 0.0f)
26	complex(const complex<float>& __z)
27	inline explicit complex(const complex<double>& __z);
28	inline explicit complex(const complex<long double>& __z)
29	value_type real()
30	value_type imag()
31	_Self& operator= (const value_type& __x)
32	_Self& operator+= (const value_type& __x)
33	_Self& operator-= (const value_type& __x)
34	_Self& operator*= (const value_type& __x)
35	_Self& operator/= (const value_type& __x)

36	static void _STLP_CALL _div(const value_type& __z1_r, const value_type& __z1_i, const value_type& __z2_r, const value_type& __z2_i, value_type& __res_r, value_type& __res_i);
37	static void _STLP_CALL _div(const value_type& __z1_r, const value_type& __z2_r, const value_type& __z2_i, vaue_type& __res_r, es_i);
38	template <class _Tp2> complex<float>& operator=(const complex<_Tp2>& __z)
39	template <class _Tp2> complex<float>& operator+= (const complex<_Tp2>& __z)
40	template <class _Tp2> complex<float>& operator-= (const complex<_Tp2>& __z)
41	template <class _Tp2> complex<float>& operator*= (const complex<_Tp2>& __z)
42	template <class _Tp2> complex<float>& operator/= (const complex<_Tp2>& __z)
43	_Self& operator=(const _Self& __z)
44	_Self& operator+= (const _Self& __z)
45	_Self& operator-= (const _Self& __z)
46	_Self& operator*= (const _Self& __z)
47	_Self& operator/= (const _Self& __z)
48	complex(value_type __x = 0.0, value_type __y = 0.0)
49	complex(const complex<double>& __z)
50	inline complex(const complex<float>& __z)
51	inline explicit complex(const complex<long double>& __z)
52	value_type real()
53	value_type imag()
54	_Self& operator= (const value_type& __x)
55	_Self& operator+= (const value_type& __x)
56	_Self& operator-= (const value_type& __x)
57	_Self& operator*= (const value_type& __x)
58	_Self& operator/= (const value_type& __x)
59	static void _STLP_CALL _div(const value_type& __z1_r, const value_type& __z1_i, const value_type& __z2_r, const value_type& __z2_i, value_type& __res_r, value_type& __res_i);
60	static void _STLP_CALL _div(const value_type& __z1_r, const value_type& __z2_r, const value_type& __z2_i, vaue_type& __res_r, es_i);
61	template <class _Tp2> complex<double>& operator=(const complex<_Tp2>& __z)
62	template <class _Tp2> complex<double>& operator+= (const complex<_Tp2>& __z)
63	template <class _Tp2> complex<double>& operator-= (const complex<_Tp2>& __z)
64	template <class _Tp2> complex<double>& operator*= (const complex<_Tp2>& __z)
65	template <class _Tp2> complex<double>& operator/= (const complex<_Tp2>& __z)
66	_Self& operator=(const _Self& __z)
67	_Self& operator+= (const _Self& __z)
68	_Self& operator-= (const _Self& __z)
69	_Self& operator*= (const _Self& __z)
70	_Self& operator/= (const _Self& __z)

71	<code>complex(value_type __x = 0.0, value_type __y = 0.0)</code>
72	<code>complex(const complex<double>& __z)</code>
73	<code>inline complex(const complex<float>& __z)</code>
74	<code>inline explicit complex(const complex<long double>& __z)</code>
75	<code>value_type real()</code>
76	<code>value_type imag()</code>
77	<code>_Self& operator= (const value_type& __x)</code>
78	<code>_Self& operator+= (const value_type& __x)</code>
79	<code>_Self& operator-= (const value_type& __x)</code>
80	<code>_Self& operator*= (const value_type& __x)</code>
81	<code>_Self& operator/= (const value_type& __x)</code>
82	<code>static void _STLP_CALL _div(const value_type& __z1_r, const value_type& __z1_i, const value_type& __z2_r, const value_type& __z2_i, value_type& __res_r, value_type& __res_i);</code>
83	<code>static void _STLP_CALL _div(const value_type& __z1_r, const value_type& __z2_r, const value_type& __z2_i, vaue_type& __res_r, es_i);</code>
84	<code>template <class _Tp2> complex<double>& operator=(const complex<_Tp2>& __z)</code>
85	<code>template <class _Tp2> complex<double>& operator+= (const complex<_Tp2>& __z)</code>
86	<code>template <class _Tp2> complex<double>& operator-= (const complex<_Tp2>& __z)</code>
87	<code>template <class _Tp2> complex<double>& operator*= (const complex<_Tp2>& __z)</code>
88	<code>template <class _Tp2> complex<double>& operator/= (const complex<_Tp2>& __z)</code>
89	<code>_Self& operator=(const _Self& __z)</code>
90	<code>_Self& operator+= (const _Self& __z)</code>
91	<code>_Self& operator-= (const _Self& __z)</code>
92	<code>_Self& operator*= (const _Self& __z)</code>
93	<code>_Self& operator/= (const _Self& __z)</code>
94	<code>inline complex<float>::complex(const complex<double>& __z)</code>
95	<code>inline complex<double>::complex(const complex<float>& __z)</code>
96	<code>inline complex<float>::complex(const complex<long double>& __z)</code>
97	<code>inline complex<double>::complex(const complex<long double>& __z)</code>
98	<code>inline complex<long double>::complex(const complex<float>& __z)</code>
99	<code>inline complex<long double>::complex(const complex<double>& __z)</code>
100	<code>template <class _Tp> inline complex<_Tp> _STLP_CALL operator+(const complex<_Tp>& __z)</code>
101	<code>template <class _Tp> inline complex<_Tp> _STLP_CALL operator-(const complex<_Tp>& __z)</code>
102	<code>template <class _Tp> inline complex<_Tp> _STLP_CALL operator+(const _Tp& __x, const complex<_Tp>& __z)</code>
103	<code>template <class _Tp> inline complex<_Tp> _STLP_CALL operator+(const complex<_Tp>& __z, const _Tp& __x)</code>
104	<code>template <class _Tp> inline complex<_Tp> _STLP_CALL operator-(const _Tp& __x, const complex<_Tp>& __z)</code>
105	<code>template <class _Tp> inline complex<_Tp> _STLP_CALL operator-(const complex<_Tp>& __z, const _Tp& __x)</code>
106	<code>template <class _Tp> inline complex<_Tp> _STLP_CALL operator*(const _Tp& __x, const complex<_Tp>& __z)</code>

107	template <class _Tp> inline complex<_Tp> _STLP_CALL operator*(const complex<_Tp>& __z, const _Tp& __x)
108	template <class _Tp> inline complex<_Tp> _STLP_CALL operator/(const _Tp& __x, const complex<_Tp>& __z)
109	template <class _Tp> inline complex<_Tp> _STLP_CALL operator/(const complex<_Tp>& __z, const _Tp& __x)
110	template <class _Tp> inline complex<_Tp> _STLP_CALL operator+(const complex<_Tp>& __z1, const complex<_Tp>& __z2)
111	template <class _Tp> inline complex<_Tp> _STLP_CALL operator-(const complex<_Tp>& __z1, const complex<_Tp>& __z2)
112	template <class _Tp> inline complex<_Tp> _STLP_CALL operator*(const complex<_Tp>& __z1, const complex<_Tp>& __z2)
113	template <class _Tp> inline complex<_Tp> _STLP_CALL operator/(const complex<_Tp>& __z1, const complex<_Tp>& __z2)
114	template <class _Tp> inline bool _STLP_CALL operator==(const complex<_Tp>& __z1, const complex<_Tp>& __z2)
115	template <class _Tp> inline bool _STLP_CALL operator==(const complex<_Tp>& __z, const _Tp& __x)
116	template <class _Tp> inline bool _STLP_CALL operator==(const _Tp& __x, const complex<_Tp>& __z)
117	template <class _Tp> inline bool _STLP_CALL operator!=(const complex<_Tp>& __z1, const complex<_Tp>& __z2)
118	template <class _Tp> inline bool _STLP_CALL operator!=(const complex<_Tp>& __z, const _Tp& __x)
119	template <class _Tp> inline bool _STLP_CALL operator!=(const _Tp& __x, const complex<_Tp>& __z)
120	template <class _Tp> inline _Tp _STLP_CALL real(const complex<_Tp>& __z)
121	template <class _Tp> inline _Tp _STLP_CALL imag(const complex<_Tp>& __z)
122	template <class _Tp> _Tp _STLP_CALL abs(const complex<_Tp>& __z)
123	template <class _Tp> _Tp _STLP_CALL arg(const complex<_Tp>& __z)
124	template <class _Tp> inline _Tp _STLP_CALL norm(const complex<_Tp>& __z)
125	template <class _Tp> inline complex<_Tp> _STLP_CALL conj(const complex<_Tp>& __z)
126	template <class _Tp> complex<_Tp> _STLP_CALL polar(const _Tp& __rho)
127	template <class _Tp> complex<_Tp> _STLP_CALL polar(const _Tp& __rho, const _Tp& __phi)
128	float _STLP_CALL abs(const complex<float>&)
129	double _STLP_CALL abs(const complex<double>&)
130	_STLP_DECLSPEC long double _STLP_CALL abs(const complex<long double>&)

131	float _STLP_CALL arg(const complex<float>&)
132	double _STLP_CALL arg(const complex<double>&)
133	_STLP_DECLSPEC long double _STLP_CALL arg(const complex<long double>&)
134	complex<float> _STLP_CALL polar(const float& __rho, const float& __phi)
135	complex<double> _STLP_CALL polar(const double& __rho, const double& __phi)
136	_STLP_DECLSPEC complex<long double> _STLP_CALL polar(const long double&, const long double&)
137	template <class _Tp> _Tp _STLP_CALL abs(const complex<_Tp>& __z)
138	template <class _Tp> _Tp _STLP_CALL arg(const complex<_Tp>& __z)
139	template <class _Tp> complex<_Tp> _STLP_CALL polar(const _Tp& __rho, const _Tp& __phi)
140	_STLP_DECLSPEC complex<float> _STLP_CALL sqrt(const complex<float>&)
141	_STLP_DECLSPEC complex<float> _STLP_CALL exp(const complex<float>&)
142	_STLP_DECLSPEC complex<float> _STLP_CALL log(const complex<float>&)
143	_STLP_DECLSPEC complex<float> _STLP_CALL log10(const complex<float>&)
144	_STLP_DECLSPEC complex<float> _STLP_CALL pow(const complex<float>&, int)
145	_STLP_DECLSPEC complex<float> _STLP_CALL pow(const complex<float>&, const float&)
146	_STLP_DECLSPEC complex<float> _STLP_CALL pow(const float&, const complex<float>&)
147	_STLP_DECLSPEC complex<float> _STLP_CALL pow(const complex<float>&, const complex<float>&)
148	_STLP_DECLSPEC complex<float> _STLP_CALL sin(const complex<float>&)
149	_STLP_DECLSPEC complex<float> _STLP_CALL cos(const complex<float>&)
150	_STLP_DECLSPEC complex<float> _STLP_CALL tan(const complex<float>&)
151	_STLP_DECLSPEC complex<float> _STLP_CALL sinh(const complex<float>&)
152	_STLP_DECLSPEC complex<float> _STLP_CALL cosh(const complex<float>&)
153	_STLP_DECLSPEC complex<float> _STLP_CALL tanh(const complex<float>&)
154	_STLP_DECLSPEC complex<double> _STLP_CALL sqrt(const complex<double>&)
155	_STLP_DECLSPEC complex<double> _STLP_CALL exp(const complex<double>&)
156	_STLP_DECLSPEC complex<double> _STLP_CALL log(const complex<double>&)
157	_STLP_DECLSPEC complex<double> _STLP_CALL log10(const complex<double>&)
158	_STLP_DECLSPEC complex<double> _STLP_CALL pow(const complex<double>&, int)
159	_STLP_DECLSPEC complex<double> _STLP_CALL pow(const complex<double>&, const double&)
160	_STLP_DECLSPEC complex<double> _STLP_CALL pow(const double&, const complex<double>&)
161	_STLP_DECLSPEC complex<double> _STLP_CALL pow(const complex<double>&, const complex<double>&)
162	_STLP_DECLSPEC complex<double> _STLP_CALL sin(const complex<double>&)
163	_STLP_DECLSPEC complex<double> _STLP_CALL cos(const complex<double>&);
164	_STLP_DECLSPEC complex<double> _STLP_CALL tan(const complex<double>&)
165	_STLP_DECLSPEC complex<double> _STLP_CALL sinh(const complex<double>&);
166	_STLP_DECLSPEC complex<double> _STLP_CALL cosh(const complex<double>&);
167	_STLP_DECLSPEC complex<double> _STLP_CALL tanh(const complex<double>&);
168	_STLP_DECLSPEC complex<long double> _STLP_CALL sqrt(const complex<long double>&)
169	_STLP_DECLSPEC complex<long double> _STLP_CALL exp(const complex<long double>&)
170	_STLP_DECLSPEC complex<long double> _STLP_CALL log(const complex<long double>&)
171	_STLP_DECLSPEC complex<long double> _STLP_CALL log10(const complex<long double>&)

172	<code>_STLP_DECLSPEC complex<long double> _STLP_CALL pow(const complex<long double>&, int)</code>
173	<code>_STLP_DECLSPEC complex<long double> _STLP_CALL pow(const complex<long double>&, const long double&)</code>
174	<code>_STLP_DECLSPEC complex<long double> _STLP_CALL pow(const long double&, const complex<long double>&)</code>
175	<code>_STLP_DECLSPEC complex<long double> _STLP_CALL pow(const complex<long double>&, const complex<long double>&)</code>
176	<code>_STLP_DECLSPEC complex<long double> _STLP_CALL sin(const complex<long double>&)</code>
177	<code>_STLP_DECLSPEC complex<long double> _STLP_CALL cos(const complex<long double>&);</code>
178	<code>_STLP_DECLSPEC complex<long double> _STLP_CALL tan(const complex<long double>&)</code>
179	<code>_STLP_DECLSPEC complex<long double> _STLP_CALL sinh(const complex<long double>&);</code>
180	<code>_STLP_DECLSPEC complex<long double> _STLP_CALL cosh(const complex<long double>&);</code>
181	<code>_STLP_DECLSPEC complex<long double> _STLP_CALL tanh(const complex<long double>&);</code>

5. <deque>

The methods supported by <deque> are listed below.

List of Methods for Class deque

1	iterator begin()
2	const_iterator begin()
3	iterator end()
4	const_iterator end()
5	reverse_iterator rbegin()
6	const_reverse_iterator rbegin()
7	reverse_iterator rend()
8	const_reverse_iterator rend()
9	reference operator[](size_type __n)
10	const_reference operator[](size_type __n)
11	void _M_range_check(size_type __n)
12	reference at(size_type __n)
13	const_reference at(size_type __n)
14	reference front()
15	const_reference front()
16	reference back()
17	const_reference back()
18	size_type size()
19	size_type max_size()
20	bool empty()
21	allocator_type get_allocator()
22	deque()
23	deque(const allocator_type& __a)
24	deque(const _Self& __x)
25	explicit deque(size_type __n)
26	deque(_InputIterator __first, _InputIterator __last, const allocator_type& __a = allocator_type())
27	deque(__move_source<_Self> src)
28	~deque ()
29	void swap(_Self& __x)
30	void _M_fill_assign(size_type __n, const_Tp& __val)
31	void assign(size_type __n, const_Tp& __val)
32	void assign(_InputIterator __first, _InputIterator __last)
34	void push_back(const value_type& __t = _Tp())
35	void push_front(const value_type& __t = _Tp())
36	void pop_back()
37	void pop_front()
38	iterator insert(iterator __pos, const value_type& __x = _Tp())
39	void insert(iterator __pos, size_type __n, const value_type& __x)
40	void insert(iterator __pos, _InputIterator __first, _InputIterator __last)
41	void clear()
42	void resize(size_type __new_size, const value_type& __x = _Tp())
43	iterator erase(iterator __pos)
44	iterator erase(iterator __first, iterator __last)
45	_Self & operator= (const _Self & __x)

46	<code>_Self & operator!= (const _Self &__x)</code>
47	<code>_Self & operator<(const _Self &__x)</code>
48	<code>_Self & operator<= (const _Self &__x)</code>
49	<code>_Self & operator==(const _Self &__x)</code>
50	<code>_Self & operator>(const _Self &__x)</code>
51	<code>_Self & operator>= (const _Self &__x)</code>

6. <exception>

The methods supported by <exception> are listed below.

List of Methods for Class exception

List of Methods for Subclass exception	
1	exception()
2	virtual ~exception()
3	const char* what()
List of Methods for Subclass bad_alloc	
4	bad_alloc ()
5	bad_alloc(const bad_alloc&)
6	bad_alloc& operator=(const bad_alloc&)
7	~bad_alloc ()
8	what()
List of Methods for Subclass bad_exception	
9	bad_exception()
10	~bad_exception()
11	what()
List of Methods for Subclass bad_cast	
12	bad_cast()
13	bad_cast(const bad_cast&)
14	bad_cast& operator=(const bad_cast&)
15	~bad_cast()
16	const char* what()
List of Methods for Subclass bad_typeid	
17	bad_typeid()
18	bad_typeid(const bad_typeid&)
19	bad_typeid& operator=(const bad_typeid&)
20	~bad_typeid()
21	const char* what()

7. <functional>

The methods supported by <functional> are listed below.

List of Methods for Class functional

List of Methods for Base Class binary_function	
1	<pre>template <class _Tp> struct not_equal_to : public binary_function<_Tp, _Tp, bool> { bool operator()(const _Tp& __x, const _Tp& __y) const</pre>
2	<pre>template <class _Tp> struct greater : public binary_function<_Tp, _Tp, bool> { bool operator()(const _Tp& __x, const _Tp& __y)</pre>
3	<pre>template <class _Tp> struct greater_equal : public binary_function<_Tp, _Tp, bool> { bool operator()(const _Tp& __x, const _Tp& __y)</pre>
4	<pre>template <class _Tp> struct less_equal : public binary_function<_Tp, _Tp, bool> { bool operator()(const _Tp& __x, const _Tp& __y)</pre>
5	<pre>template <class _Tp> struct divides : public binary_function<_Tp, _Tp, _Tp> { _Tp operator()(const _Tp& __x, const _Tp& __y)</pre>
6	<pre>template <class _Tp> struct modulus : public binary_function<_Tp, _Tp, _Tp> { _Tp operator()(const _Tp& __x, const _Tp& __y)</pre>
7	<pre>template <class _Tp> struct logical_and : public binary_function<_Tp, _Tp, bool> { bool operator()(const _Tp& __x, const _Tp& __y)</pre>
8	<pre>template <class _Tp> struct logical_or : public binary_function<_Tp, _Tp, bool> { bool operator()(const _Tp& __x, const _Tp& __y) const</pre>
List of Methods for Subclass unary_function	
9	<pre>template <class _Tp> struct negate : public unary_function<_Tp, _Tp> { _Tp operator()(const _Tp& __x)</pre>
10	<pre>template <class _Tp> struct logical_not : public unary_function<_Tp, bool> { bool operator()(const _Tp& __x)</pre>
List of Methods for Subclass unary_negate	
11	<pre>explicit unary_negate(const _Predicate& __x)</pre>
12	<pre>bool operator()(_ArgParamType __x)</pre>
13	<pre>template <class _Predicate> inline unary_negate<_Predicate> not1(const _Predicate& __pred)</pre>
List of Methods for Subclass binary_negate	
14	<pre>explicit binary_negate(const _Predicate& __x)</pre>
15	<pre>bool operator()(_FstArgParamType __x, _SndArgParamType __y)</pre>
16	<pre>binary_negate<_Predicate> not2(const _Predicate& __pred)</pre>

List of Methods for Subclass binder1st

- 17 binder1st(const _Operation& __x, _ValueParamType __y)
- 18 result_type operator()(_ConstArgParamType __x)
- 19 result_type operator()(_ArgParamType __x)

List of Methods for Subclass binder2nd

- 20 binder2nd(const _Operation& __x, _ValueParamType __y)
- 21 result_type operator()(_ConstArgParamType __x)
- 22 result_type operator()(_ArgParamType __x)

List of Methods for Subclass unary_compose

- 23 unary_compose(const _Operation1& __x, const _Operation2& __y)
- 24 result_type operator()(_ArgParamType __x)

List of Methods for Subclass binary_compose

- 25 binary_compose(const _Operation1& __x, const _Operation2& __y, const _Operation3& __z)
- 26 result_type operator()(_ArgParamType __x)
- 27 template <class _Operation1, class _Operation2, class _Operation3>
 inline binary_compose<_Operation1, _Operation2, _Operation3>
 compose2(const _Operation1& __fn1, const _Operation2& __fn2,
 const _Operation3& __fn3)

List of Methods for Subclass constant_void_fun

- 28 _Constant_void_fun(const result_type& __v)
- 29 const result_type& operator()
- 30 template <class _Result>
 struct constant_void_fun : public _STLP_PRIV _Constant_void_fun<_Result> {
 constant_void_fun(const _Result& __v)
- 31 template <class _Result, _STLP_DFL_TMPL_PARAM(_Argument , _Result) >
 struct constant_unary_fun : public _STLP_PRIV _Constant_unary_fun<_Result, _Argument> {
 constant_unary_fun(const _Result& __v)
- 32 template <class _Result, _STLP_DFL_TMPL_PARAM(_Arg1 , _Result),
 _STLP_DFL_TMPL_PARAM(_Arg2 , _Arg1) >
 struct constant_binary_fun
 : public _STLP_PRIV _Constant_binary_fun<_Result, _Arg1, _Arg2> {
 constant_binary_fun(const _Result& __v)
- 33 template <class _Result>
 inline constant_void_fun<_Result> constant0(const _Result& __val)
- 34 template <class _Result>
 inline constant_unary_fun<_Result, _Result> constant1(const _Result& __val)
- 35 template <class _Result>
 inline constant_binary_fun<_Result, _Result, _Result>
 constant2(const _Result& __val)

List of Methods for Subclass subtractive_rng

- 36 _STLP_UINT32_T operator()(_STLP_UINT32_T __limit)
- 37 void _M_initialize(_STLP_UINT32_T __seed)
- 38 subtractive_rng(unsigned int __seed)

8. <iterator>

The methods supported by <iterator> are listed below.

List of Methods for Class iterator

List of Methods for Subclass reverse_iterator	
1	reverse_iterator()
2	explicit reverse_iterator(iterator_type __x)
3	reverse_iterator(const _Self& __x)
4	template <class _Iter> reverse_iterator(const reverse_iterator<_Iter>& __x)
5	_Self& operator = (const _Self& __x)
6	_Self& operator = (const reverse_iterator<_Iter>& __x)
7	iterator_type base()
8	reference operator*()
9	_Self& operator++()
10	_Self operator++(int)
11	_Self& operator--()
12	_Self operator--(int)
13	_Self operator+(difference_type __n)
14	_Self& operator+=(difference_type __n)
15	_Self operator-(difference_type __n)
16	_Self& operator-=(difference_type __n)
17	reference operator[](difference_type __n)
18	template <class _Iterator> inline bool _STLP_CALL operator==(const reverse_iterator<_Iterator>& __x, const reverse_iterator<_Iterator>& __y)
19	template <class _Iterator> inline bool _STLP_CALL operator<(const reverse_iterator<_Iterator>& __x, const reverse_iterator<_Iterator>& __y)
20	template <class _Iterator> inline bool _STLP_CALL operator!=(const reverse_iterator<_Iterator>& __x, const reverse_iterator<_Iterator>& __y)
21	template <class _Iterator> inline bool _STLP_CALL operator>(const reverse_iterator<_Iterator>& __x, const reverse_iterator<_Iterator>& __y)
22	template <class _Iterator> inline bool _STLP_CALL operator<=(const reverse_iterator<_Iterator>& __x, const reverse_iterator<_Iterator>& __y)
23	template <class _Iterator> inline bool _STLP_CALL operator>=(const reverse_iterator<_Iterator>& __x, const reverse_iterator<_Iterator>& __y)
List of Methods for Subclass back_insert_iterator	
24	explicit back_insert_iterator(_Container& __x)
25	_Self& operator=(const _Self& __other)
26	_Self& operator=(const typename _Container::value_type& __val)
27	_Self& operator*()
28	_Self& operator++()
29	_Self operator++(int)

List of Methods for Subclass front_insert_iterator

30	explicit front_insert_iterator(_Container& __x)
31	_Self& operator=(const _Self& __other)
32	_Self& operator=(const typename _Container::value_type& __val)
33	_Self& operator*()
34	_Self& operator++()
35	_Self operator++(int)

List of Methods for Subclass insert_iterator

36	insert_iterator(_Container& __x, typename _Container::iterator __i)
37	_Self& operator=(const _Self& __other)
38	_Self& operator=(const typename _Container::value_type& __val)
39	_Self& operator*()
40	_Self& operator++()
41	_Self operator++(int)

9. <limits>

The methods supported by <limits> are listed below.

List of Methods for Class limits

1	static __number (_STLP_CALL min)()
2	static __number (_STLP_CALL max)()
3	_STLP_STATIC_CONSTANT(int, digits = 0)
4	_STLP_STATIC_CONSTANT(int, digits10 = 0)
5	_STLP_STATIC_CONSTANT(int, radix = 0)
6	_STLP_STATIC_CONSTANT(int, min_exponent = 0)
7	_STLP_STATIC_CONSTANT(int, min_exponent10 = 0)
8	_STLP_STATIC_CONSTANT(int, max_exponent = 0)
9	_STLP_STATIC_CONSTANT(int, max_exponent10 = 0)
10	_STLP_STATIC_CONSTANT(float_denorm_style, has_denorm = denorm_absent)
11	_STLP_STATIC_CONSTANT(float_round_style, round_style = round_toward_zero)
12	_STLP_STATIC_CONSTANT(bool, is_specialized = false)
13	_STLP_STATIC_CONSTANT(bool, is_signed = false)
14	_STLP_STATIC_CONSTANT(bool, is_integer = false)
15	_STLP_STATIC_CONSTANT(bool, is_exact = false)
16	_STLP_STATIC_CONSTANT(bool, has_infinity = false)
17	_STLP_STATIC_CONSTANT(bool, has_quiet_NaN = false)
18	_STLP_STATIC_CONSTANT(bool, has_signaling_NaN = false)
19	_STLP_STATIC_CONSTANT(bool, has_denorm_loss = false)
20	_STLP_STATIC_CONSTANT(bool, is_iec559 = false)
21	_STLP_STATIC_CONSTANT(bool, is_bounded = false)
22	_STLP_STATIC_CONSTANT(bool, is_modulo = false)
23	_STLP_STATIC_CONSTANT(bool, traps = false)
24	_STLP_STATIC_CONSTANT(bool, tinyness_before = false)
25	__number _STLP_CALL epsilon()
26	__number _STLP_CALL round_error()
27	__number _STLP_CALL infinity()
28	__number _STLP_CALL quiet_NaN()
29	__number _STLP_CALL signaling_NaN()
30	__number _STLP_CALL denorm_min()

10. <list>

The methods supported by <list> are listed below.

List of Methods for Class list

1	list(size_type __n, const_reference __val = _STLP_DEFAULT_CONSTRUCTED(value_type), const allocator_type& __a = allocator_type())
2	list(_InputIterator __first, _InputIterator __last, const allocator_type& __a _STLP_ALLOCATOR_TYPE_DFL)
3	list(const allocator_type& __a = allocator_type())
4	list(const _Self& __x)
5	list(__move_source<_Self> src)
6	void clear()
7	bool empty()
8	iterator begin()
9	const_iterator begin()
10	iterator end()
11	const_iterator end()
12	reverse_iterator rbegin()
13	const_reverse_iterator rbegin()
14	reverse_iterator rend()
15	const_reverse_iterator rend()
16	size_type size()
17	size_type max_size()
18	reference front()
19	const_reference front()
20	reference back()
21	const_reference back()
22	void swap(_Self& __x)
23	iterator insert(iterator __pos, const_reference __x = value_type())
24	void insert(iterator __pos, _InputIterator __first, _InputIterator __last)
25	void insert(iterator __pos, size_type __n, const_reference __x)
26	void push_front(const_reference __x)
27	void push_back (const_reference __x)
28	iterator erase(iterator __pos)
29	iterator erase(iterator __first, iterator __last)
30	void resize(size_type __new_size, const_reference __x = value_type())
31	void pop_front()
32	void pop_back()
33	void assign(size_type __n, const_reference __val)
34	void assign(_InputIterator __first, _InputIterator __last)
35	void splice(iterator __pos, _Self& __x)
36	void splice(iterator __pos, _Self& __x, iterator __i)
37	void splice(iterator __pos, _Self& __x, iterator __first, iterator __last)
38	void remove(const_reference __val)
39	void unique()
40	void merge(_Self& __x)

41	void reverse()
42	void sort()
43	_Self& operator=
44	bool _STLP_CALL operator==
45	bool _STLP_CALL operator!=
46	bool _STLP_CALL operator<
47	bool _STLP_CALL operator<=
48	bool _STLP_CALL operator>
49	bool _STLP_CALL operator>=
50	allocator_type get_allocator()
51	void _M_fill_assign(size_type __n, const_reference __val)

11. <map>

The methods supported by <map> are listed below.

List of Methods for Class map

1	map()
2	map(const _Compare& __comp, const allocator_type& __a = allocator_type())
3	map(_InputIterator __first, _InputIterator __last)
4	map(_InputIterator __first, _InputIterator __last, const _Compare& __comp, const allocator_type& __a _STLP_ALLOCATOR_TYPE_DFL)
5	map(const _Self& __x)
6	map(__move_source<_Self> src)
7	key_compare key_comp()
8	value_compare value_comp()
9	allocator_type get_allocator()
10	iterator begin()
11	const_iterator begin()
12	iterator end()
13	const_iterator end()
14	reverse_iterator rbegin()
15	const_reverse_iterator rbegin()
16	reverse_iterator rend()
17	const_reverse_iterator rend()
18	bool empty()
19	size_type size()
20	size_type max_size()
21	void swap(_Self& __x)
22	pair<iterator,bool> insert(const value_type& __x)
23	iterator insert(iterator __pos, const value_type& __x)
24	void insert(_InputIterator __first, _InputIterator __last)
25	void erase(iterator __pos)
26	size_type erase(const key_type& __x)
27	void erase(iterator __first, iterator __last)
28	void clear()
29	iterator find(const _KT& __x)
30	const_iterator find(const _KT& __x)
31	size_type count(const _KT& __x)
32	iterator lower_bound(const _KT& __x)
33	const_iterator lower_bound(const _KT& __x)
34	iterator upper_bound(const _KT& __x)
35	const_iterator upper_bound(const _KT& __x)
36	pair<iterator,iterator> equal_range(const _KT& __x)
37	pair<const_iterator,const_iterator> equal_range(const _KT& __x)
38	_Self& operator= (const _Self& __x)
39	_Tp& operator[] (const _KT& __k)
40	bool _STLP_CALL operator!= (const map<_Key,_Tp,_Compare,_Alloc>& __x, const map<_Key,_Tp,_Compare,_Alloc>& __y)

41	<code>bool _STLP_CALL operator< (const map<_Key,_Tp,_Compare,_Alloc>& __x, const map<_Key,_Tp,_Compare,_Alloc>&__y)</code>
42	<code>bool _STLP_CALL operator<= (const map<_Key,_Tp,_Compare,_Alloc>& __x, const map<_Key,_Tp,_Compare,_Alloc>&__y)</code>
43	<code>bool _STLP_CALL operator== (const map<_Key,_Tp,_Compare,_Alloc>& __x, const map<_Key,_Tp,_Compare,_Alloc>&__y)</code>
44	<code>bool _STLP_CALL operator> (const map<_Key,_Tp,_Compare,_Alloc>& __x, const map<_Key,_Tp,_Compare,_Alloc>&__y)</code>
45	<code>bool _STLP_CALL operator>= (const map<_Key,_Tp,_Compare,_Alloc>& __x, const map<_Key,_Tp,_Compare,_Alloc>&__y)</code>

List of Methods for Class multimap

1	multimap()
2	multimap(const _Compare& __comp, const allocator_type& __a = allocator_type())
3	multimap(_InputIterator __first, _InputIterator __last)
4	multimap(_InputIterator __first, _InputIterator __last, const _Compare& __comp, const allocator_type& __a = STLP_ALLOCATOR_TYPE_DFL)
5	multimap(const _Self& __x)
6	multimap(__move_source<_Self> src)
7	key_compare key_comp()
8	value_compare value_comp()
9	allocator_type get_allocator()
10	iterator begin()
11	const_iterator begin()
12	iterator end()
13	const_iterator end()
14	reverse_iterator rbegin()
15	const_reverse_iterator rbegin()
16	reverse_iterator rend()
17	const_reverse_iterator rend()
18	bool empty()
19	size_type size()
20	size_type max_size()
21	void swap(_Self& __x)
22	iterator insert(const value_type& __x)
23	iterator insert(iterator __pos, const value_type& __x)
24	void insert(_InputIterator __first, _InputIterator __last)
25	void erase(iterator __pos)
26	size_type erase(const key_type& __x)
27	void erase(iterator __first, iterator __last)
28	void clear()
29	iterator find(const _KT& __x)
30	const_iterator find(const _KT& __x)
31	size_type count(const _KT& __x)
32	iterator lower_bound(const _KT& __x)
33	const_iterator lower_bound(const _KT& __x)
34	iterator upper_bound(const _KT& __x)
35	const_iterator upper_bound(const _KT& __x)
36	pair<iterator,iterator> equal_range(const _KT& __x)
37	pair<const_iterator,const_iterator> equal_range(const _KT& __x)
38	_Self& operator= (const _Self& __x)
39	bool _STLP_CALL operator!= (const multimap<_Key,_Tp,_Compare,_Alloc>& __x, const multimap<_Key,_Tp,_Compare,_Alloc>& __y)
40	bool _STLP_CALL operator< (const multimap<_Key,_Tp,_Compare,_Alloc>& __x, const multimap<_Key,_Tp,_Compare,_Alloc>& __y)
41	bool _STLP_CALL operator<= (const multimap<_Key,_Tp,_Compare,_Alloc>& __x, const multimap<_Key,_Tp,_Compare,_Alloc>& __y)

42	bool _STLP_CALL operator==(const multimap<_Key,_Tp,_Compare,_Alloc>& __x, const multimap<_Key,_Tp,_Compare,_Alloc>& __y)
43	bool _STLP_CALL operator>(const multimap<_Key,_Tp,_Compare,_Alloc>& __x, const multimap<_Key,_Tp,_Compare,_Alloc>& __y)
44	bool _STLP_CALL operator>=(const multimap<_Key,_Tp,_Compare,_Alloc>& __x, const multimap<_Key,_Tp,_Compare,_Alloc>& __y)

12. <memory>

The methods supported by <memory> are listed below.

List of Methods for Class allocator

1	allocator()
2	allocator(const allocator<_Tp1>&)
3	allocator(__move_source<allocator<_Tp> > src)
4	~allocator()
5	pointer address(reference __x)
6	const_pointer address(const_reference __x)
7	_Tp* allocate(size_type __n, const void* = 0)
8	void deallocate(pointer __p, size_type __n)
9	void deallocate(pointer __p) const
10	size_type max_size() const
11	void construct(pointer __p, const_reference __val)
12	void destroy(pointer __p)

List of Methods for Class auto_ptr

1	_Tp* release()
2	void reset(_Tp* __px = 0)
3	_Tp* get()
4	auto_ptr(_Tp* __px = 0)
5	auto_ptr(_Self& __r)
6	auto_ptr(auto_ptr_ref<_Tp> __r)
7	operator auto_ptr_ref<_Tp1>()
8	_Tp* operator->()
9	_Tp* operator*()
10	_Self& operator= (_Self& __r)
11	_Self& operator= (auto_ptr_ref<_Tp> __r)
12	operator auto_ptr<_Tp1>()

List of Methods for Class auto_ptr_ref

1	auto_ptr_ref(__ptr_base& __r, _Tp* __p)
2	_Tp* release()

List of Methods for Class raw_storage_iterator

1	raw_storage_iterator(_ForwardIterator __x)
2	raw_storage_iterator<_ForwardIterator, _Tp>& operator*()
3	raw_storage_iterator<_ForwardIterator, _Tp>& operator=(const _Tp& __element)
4	raw_storage_iterator<_ForwardIterator, _Tp>& operator++()
5	raw_storage_iterator<_ForwardIterator, _Tp> operator++(int)

13. <numeric>

The methods supported by <numeric> are listed below.

List of Methods for Class numeric

1	<code>_Tp accumulate(_InputIterator __first, _InputIterator __last, _Tp _Init)</code>
2	<code>_Tp accumulate(_InputIterator __first, _InputIterator __last, _Tp _Init, _BinaryOperation __binary_op)</code>
3	<code>_Tp inner_product(_InputIterator1 __first1, _InputIterator1 __last1, _InputIterator2 __first2, _Tp _Init)</code>
4	<code>_Tp inner_product(_InputIterator1 __first1, _InputIterator1 __last1, _InputIterator2 __first2, _Tp _Init, _BinaryOperation1 __binary_op1, _BinaryOperation2 __binary_op2)</code>
5	<code>_OutputIterator partial_sum(_InputIterator __first, _InputIterator __last, _OutputIterator __result)</code>
6	<code>_OutputIterator partial_sum(_InputIterator __first, _InputIterator __last, _OutputIterator __result, _BinaryOperation __binary_op)</code>
7	<code>_OutputIterator adjacent_difference(_InputIterator __first, _InputIterator __last, _OutputIterator __result)</code>
8	<code>_OutputIterator adjacent_difference(_InputIterator __first, _InputIterator __last, _OutputIterator __result, _BinaryOperation __binary_op)</code>
9	<code>_Tp power(_Tp __x, _Integer __n, _MonoidOperation __opr)</code>
10	<code>_Tp power(_Tp __x, _Integer __n)</code>
11	<code>void iota(_ForwardIterator __first, _ForwardIterator __last, _Tp __val)</code>

14. <queue>

The methods supported by <queue> are listed below.

List of Methods for Class queue

1	queue()
2	queue(const _Sequence& __c)
3	queue(__move_source<_Self> src)
4	bool empty()
5	size_type size()
6	reference front()
7	const_reference front()
8	reference back()
9	const_reference back()
10	void push(const value_type& __x)
11	void pop()
12	const _Sequence& _Get_s()
13	bool _STLP_CALL operator==(const queue<_Tp>& __x, const queue<_Tp>& __y)
14	bool _STLP_CALL operator<(const queue<_Tp>& __x, const queue<_Tp>& __y)
15	bool _STLP_CALL operator>(const queue<_Tp>& __x, const queue<_Tp>& __y)
16	bool _STLP_CALL operator<=(const queue<_Tp>& __x, const queue<_Tp>& __y)
17	bool _STLP_CALL operator>=(const queue<_Tp>& __x, const queue<_Tp>& __y)
18	bool _STLP_CALL operator!=(const queue<_Tp>& __x, const queue<_Tp>& __y)

List of Methods for Class priority_queue

1	priority_queue()
2	priority_queue(const _Compare& __x)
3	priority_queue(const _Compare& __x, const _Sequence& __s)
4	priority_queue(__move_source<_Self> src)
5	priority_queue(_InputIterator __first, _InputIterator __last)
6	priority_queue(_InputIterator __first, _InputIterator __last, const _Compare& __x)
7	priority_queue(_InputIterator __first, _InputIterator __last, const _Compare& __x, const _Sequence& __s)
8	bool empty()
9	size_type size()
10	const_reference top()
11	void push(const value_type& __x)
12	void pop()

15. <set>

The methods supported by <set> are listed below.

List of Methods for Class set

1	set(const _Compare& __comp = _Compare(),const allocator_type& __a = allocator_type())
2	set(_InputIterator __first, _InputIterator __last)
3	set(_InputIterator __first, _InputIterator __last, const _Compare& __comp,const allocator_type& __a = allocator_type())
4	set(const _Self& __x)
5	set(__move_source<_Self> src)
6	key_compare key_comp()
7	value_compare value_comp()
8	allocator_type get_allocator()
9	iterator begin()
10	const_iterator begin()
11	iterator end()
12	const_iterator end()
13	reverse_iterator rbegin()
14	const_reverse_iterator rbegin()
15	reverse_iterator rend()
16	const_reverse_iterator rend()
17	bool empty()
18	size_type size()
19	size_type max_size()
20	swap(_Self& __x)
21	pair<iterator,bool> insert(const value_type& __x)
22	iterator insert(iterator __pos, const value_type& __x)
23	void insert(_InputIterator __first, _InputIterator __last)
24	void erase(iterator __pos)
25	size_type erase(const key_type& __x)
26	void erase(iterator __first, iterator __last)
27	clear()
28	const_iterator find(const _KT& __x)
29	iterator find(const _KT& __x)
30	size_type count(const _KT& __x)
31	iterator lower_bound(const _KT& __x)
32	const_iterator lower_bound(const _KT& __x)
33	iterator upper_bound(const _KT& __x)
34	const_iterator upper_bound(const _KT& __x)
35	pair<iterator, iterator> equal_range(const _KT& __x)
36	pair<const_iterator, const_iterator> equal_range(const _KT& __x)
37	_Self& operator=(const _Self& __x)
38	bool _STLP_CALL operator!=(const set<_Key,_Compare,_Alloc>& __x, const set<_Key,_Compare,_Alloc>& __y)
39	bool _STLP_CALL operator< (const set<_Key,_Compare,_Alloc>& __x, const set<_Key,_Compare,_Alloc>& __y)
40	bool _STLP_CALL operator<= (const set<_Key,_Compare,_Alloc>& __x, const set<_Key,_Compare,_Alloc>& __y)

41	<code>bool _STLP_CALL operator==(const set<_Key,_Compare,_Alloc>& __x, const set<_Key,_Compare,_Alloc>& __y)</code>
42	<code>bool _STLP_CALL operator>(const set<_Key,_Compare,_Alloc>& __x, const set<_Key,_Compare,_Alloc>& __y)</code>
43	<code>bool _STLP_CALL operator>=(const set<_Key,_Compare,_Alloc>& __x, const set<_Key,_Compare,_Alloc>& __y)</code>

List of Methods for Class multiset

1	multiset(const _Compare& __comp = _Compare(),const allocator_type& __a = allocator_type())
2	multiset(_InputIterator __first, _InputIterator __last)
3	multiset(_InputIterator __first, _InputIterator __last,const _Compare& __comp,const allocator_type& __a = allocator_type())
4	multiset(const _Self& __x)
5	multiset(__move_source<_Self> src)
6	key_compare key_comp()
7	value_compare value_comp()
8	allocator_type get_allocator()
9	iterator begin()
10	const_iterator begin()
11	iterator end()
12	const_iterator end()
13	reverse_iterator rbegin()
14	const_reverse_iterator rbegin()
15	reverse_iterator rend()
16	const_reverse_iterator rend()
17	bool empty()
18	size_type size()
19	size_type max_size()
20	void swap(_Self& __x)
21	iterator insert(const value_type& __x)
22	iterator insert(iterator __pos, const value_type& __x)
23	void insert(_InputIterator __first, _InputIterator __last)
24	void erase(iterator __pos)
25	size_type erase(const key_type& __x)
26	void erase(iterator __first, iterator __last)
27	void clear()
28	iterator find(const _KT& __x)
29	const_iterator find(const _KT& __x)
30	size_type count(const _KT& __x)
31	iterator lower_bound(const _KT& __x)
32	const_iterator lower_bound(const _KT& __x)
33	iterator upper_bound(const _KT& __x)
34	const_iterator upper_bound(const _KT& __x)
35	pair<iterator, iterator> equal_range(const _KT& __x)
36	pair<const_iterator, const_iterator> equal_range(const _KT& __x)
37	_Self& operator=(const _Self& __x)
38	bool _STLP_CALL operator!=(const multiset<_Key,_Compare,_Alloc>& __x, const multiset<_Key,_Compare,_Alloc>& __y)
39	bool _STLP_CALL operator< (const multiset<_Key,_Compare,_Alloc>& __x, const multiset<_Key,_Compare,_Alloc>& __y)
40	bool _STLP_CALL operator<= (const multiset<_Key,_Compare,_Alloc>& __x, const multiset<_Key,_Compare,_Alloc>& __y)
41	bool _STLP_CALL operator== (const multiset<_Key,_Compare,_Alloc>& __x, const multiset<_Key,_Compare,_Alloc>& __y)

42	<code>bool _STLP_CALL operator> (const multiset<_Key,_Compare,_Alloc>& __x, const multiset<_Key,_Compare,_Alloc>& __y)</code>
43	<code>bool _STLP_CALL operator>= (const multiset<_Key,_Compare,_Alloc>& __x, const multiset<_Key,_Compare,_Alloc>& __y)</code>

16. <stack>

The methods supported by <stack> are listed below.

List of Methods for Class stack

1	stack()
2	stack(const _Sequence& __s)
3	stack(__move_source<_Self> src)
4	bool empty()
5	size_type size()
6	reference top()
7	const_reference top()
8	void push(const value_type& __x)
9	void pop()
10	const _Sequence& _Get_s()
11	bool _STLP_CALL operator!= (const stack<_Tp>& __x, const stack<_Tp>& __y)
12	bool _STLP_CALL operator< (const stack<_Tp>& __x, const stack<_Tp>& __y)
13	bool _STLP_CALL operator<= (const stack<_Tp>& __x, const stack<_Tp>& __y)
14	bool _STLP_CALL operator== (const stack<_Tp>& __x, const stack<_Tp>& __y)
15	bool _STLP_CALL operator> (const stack<_Tp>& __x, const stack<_Tp>& __y)
16	bool _STLP_CALL operator>= (const stack<_Tp>& __x, const stack<_Tp>& __y)

17. <stdexcept>

The methods supported by <stdexcept> are listed below.

List of Methods for Class stdexcept

1	logic_error(const string& __s)
2	runtime_error(const string& __s)
3	domain_error(const string& __arg)
4	invalid_argument(const string& __arg)
5	length_error(const string& __arg)
6	out_of_range(const string& __arg)
7	range_error(const string& __arg)
8	overflow_error(const string& __arg)
9	underflow_error(const string& __arg)

18. <string>

The methods supported by <string> are listed below.

List of Methods for Class string

1	explicit basic_string(const allocator_type& __a = allocator_type())
2	basic_string(_Reserve_t, size_t __n, const allocator_type& __a = allocator_type())
3	basic_string(const _Self&)
4	basic_string(const _Self& __s, size_type __pos, size_type __n = npos, const allocator_type& __a = allocator_type())
5	basic_string(const _CharT* __s, size_type __n, const allocator_type& __a = allocator_type())
6	basic_string(const _CharT* __s, const allocator_type& __a = allocator_type());
7	basic_string(size_type __n, _CharT __c, const allocator_type& __a = allocator_type())
8	basic_string(__move_source<_Self> src)
9	basic_string(_InputIter __f, _InputIter __l, const allocator_type & __a = allocator_type())
10	_Self& operator=(const _Self& __s)
11	_Self& operator=(const _CharT* __s)
12	_Self& operator=(_CharT __c)
13	iterator begin()
14	const_iterator begin()
15	iterator end()
16	const_iterator end()
17	reverse_iterator rbegin()
18	const_reverse_iterator rbegin()
19	reverse_iterator rend()
20	const_reverse_iterator rend()
21	size_type size()
22	size_type length()
23	size_type max_size()
24	void resize(size_type __n, _CharT __c)
25	void resize(size_type __n)
26	void reserve(size_type = 0)
27	size_type capacity()
28	void clear()
29	bool empty()
30	const_reference operator[](size_type __n)
31	reference operator[](size_type __n)
32	const_reference at(size_type __n)
33	reference at(size_type __n)
34	_Self& operator+=(const _Self& __s)
35	_Self& operator+=(const _CharT* __s)
36	_Self& operator+=(_CharT __c)
37	_Self& append(_InputIter __first, _InputIter __last)
38	_Self& append(const _Self& __s)
39	_Self& append(const _CharT* __s, size_type __n)
40	_Self& append(const _CharT* __s)

41	void push_back(_CharT __c)
42	void pop_back()
43	_Self& assign(const _Self& __s)
44	_Self& assign(const _Self& __s, size_type __pos, size_type __n)
45	_Self& assign(const _CharT* __s, size_type __n)
46	_Self& assign(const _CharT* __s)
47	_Self& assign(size_type __n, _CharT __c)
48	_Self& assign(_Inputtler __first, _Inputtler __last)
49	_Self& insert(size_type __pos, const _Self& __s)
50	_Self& insert(size_type __pos, const _Self& __s, size_type __beg, size_type __n)
51	_Self& insert(size_type __pos, const _CharT* __s, size_type __n)
52	_Self& insert(size_type __pos, const _CharT* __s)
53	_Self& insert(size_type __pos, size_type __n, _CharT __c)
54	iterator insert(iterator __p, _CharT __c)
55	void insert(iterator __p, size_t __n, _CharT __c)
56	void insert(iterator __p, _Inputtler __first, _Inputtler __last)
57	void insert(iterator __p, const _CharT* __f, const _CharT* __l)
58	_Self& erase(size_type __pos = 0, size_type __n = npos)
59	iterator erase(iterator __pos)
60	iterator erase(iterator __first, iterator __last)
61	_Self& replace(size_type __pos, size_type __n, const _Self& __s)
62	_Self& replace(size_type __pos1, size_type __n1, const _Self& __s, size_type __pos2, size_type __n2)
63	_Self& replace(size_type __pos, size_type __n1, const _CharT* __s, size_type __n2)
64	_Self& replace(size_type __pos, size_type __n1, const _CharT* __s)
65	_Self& replace(size_type __pos, size_type __n1, size_type __n2, _CharT __c)
66	_Self& replace(iterator __first, iterator __last, const _Self& __s)
67	_Self& replace(iterator __first, iterator __last, const _CharT* __s)
68	_Self& replace(iterator __first, iterator __last, _Inputtler __f, _Inputtler __l)
69	Self& replace(iterator __first, iterator __last, const _CharT* __f, const _CharT* __l)
70	size_type copy(_CharT* __s, size_type __n, size_type __pos = 0)
71	void swap(_Self& __s)
72	const _CharT* c_str()
73	const _CharT* data()
74	size_type find(const _Self& __s, size_type __pos = 0)
75	size_type find(const _CharT* __s, size_type __pos = 0)
76	size_type find(const _CharT* __s, size_type __pos, size_type __n)
77	size_type find(_CharT __c)
78	size_type find(_CharT __c, size_type __pos /* = 0 */)
79	size_type rfind(const _Self& __s, size_type __pos = npos)
80	size_type rfind(const _CharT* __s, size_type __pos = npos)
81	size_type rfind(const _CharT* __s, size_type __pos, size_type __n)
82	size_type rfind(_CharT __c, size_type __pos = npos)
83	size_type find_first_of(const _Self& __s, size_type __pos = 0)
84	size_type find_first_of(const _CharT* __s, size_type __pos = 0)
85	size_type find_first_of(const _CharT* __s, size_type __pos, size_type __n)
86	size_type find_first_of(_CharT __c, size_type __pos = 0)
87	size_type find_last_of(const _Self& __s, size_type __pos = npos)
88	size_type find_last_of(const _CharT* __s, size_type __pos = npos)

89	size_type find_last_of(const _CharT* __s, size_type __pos, size_type __n)
90	size_type find_last_of(_CharT __c, size_type __pos = npos)
91	size_type find_first_not_of(const _Self& __s, size_type __pos = 0)
92	size_type find_first_not_of(const _CharT* __s, size_type __pos = 0)
93	size_type find_first_not_of(const _CharT* __s, size_type __pos, size_type __n)
94	size_type find_first_not_of(_CharT __c, size_type __pos = 0)
95	size_type find_last_not_of(const _Self& __s, size_type __pos = npos)
96	size_type find_last_not_of(const _CharT* __s, size_type __pos = npos)
97	size_type find_last_not_of(const _CharT* __s, size_type __pos, size_type __n)
98	size_type find_last_not_of(_CharT __c, size_type __pos = npos)
99	_Self substr(size_type __pos = 0, size_type __n = npos)
100	int compare(const _Self& __s)
101	int compare(size_type __pos1, size_type __n1, const _Self& __s)
102	int compare(size_type __pos1, size_type __n1, const _Self& __s, size_type __pos2, size_type __n2)
103	int compare(const _CharT* __s)
104	int compare(size_type __pos1, size_type __n1, const _CharT* __s)
105	int compare(size_type __pos1, size_type __n1, const _CharT* __s, size_type __n2)
106	static int _M_compare(const _CharT* __f1, const _CharT* __l1, const _CharT* __f2, const _CharT* __l2)
107	basic_string<_CharT, _Traits, _Alloc> operator+(const basic_string<_CharT, _Traits, _Alloc>& __s, const basic_string<_CharT, _Traits, _Alloc>& __y)
108	basic_string<_CharT, _Traits, _Alloc> operator+(const _CharT* __s, const basic_string<_CharT, _Traits, _Alloc>& __y)
109	basic_string<_CharT, _Traits, _Alloc> operator+(_CharT __c, const basic_string<_CharT, _Traits, _Alloc>& __y)
110	basic_string<_CharT, _Traits, _Alloc> operator+(const basic_string<_CharT, _Traits, _Alloc>& __x, const _CharT* __s)
111	basic_string<_CharT, _Traits, _Alloc> operator+(const basic_string<_CharT, _Traits, _Alloc>& __x, const _CharT __c)
112	bool _STLP_CALL operator==(const _CharT* __s, const basic_string<_CharT, _Traits, _Alloc>& __y)
113	bool _STLP_CALL operator==(const basic_string<_CharT, _Traits, _Alloc>& __x, const _CharT* __s)
114	bool _STLP_CALL operator<(const basic_string<_CharT, _Traits, _Alloc>& __x, const basic_string<_CharT, _Traits, _Alloc>& __y)
115	bool _STLP_CALL operator<(const basic_string<_CharT, _Traits, _Alloc>& __x, const _CharT* __s)
116	bool _STLP_CALL operator!=(const basic_string<_CharT, _Traits, _Alloc>& __x, const basic_string<_CharT, _Traits, _Alloc>& __y)
117	bool _STLP_CALL operator>(const basic_string<_CharT, _Traits, _Alloc>& __x, const basic_string<_CharT, _Traits, _Alloc>& __y)
118	bool _STLP_CALL operator<=(const basic_string<_CharT, _Traits, _Alloc>& __x, const basic_string<_CharT, _Traits, _Alloc>& __y)
119	bool _STLP_CALL operator>=(const basic_string<_CharT, _Traits, _Alloc>& __x, const basic_string<_CharT, _Traits, _Alloc>& __y)
120	bool _STLP_CALL operator!=(const _CharT* __s, const basic_string<_CharT, _Traits, _Alloc>& __y)
121	bool _STLP_CALL operator!=(const basic_string<_CharT, _Traits, _Alloc>& __x, const _CharT* __s)
122	bool _STLP_CALL operator>(const _CharT* __s, const basic_string<_CharT, _Traits, _Alloc>& __y)
123	bool _STLP_CALL operator>(const basic_string<_CharT, _Traits, _Alloc>& __x, const _CharT* __s)

124	bool _STLP_CALL operator<=(const _CharT* __s, const basic_string<_CharT,_Traits,_Alloc>& __y)
125	bool _STLP_CALL operator<=(const basic_string<_CharT,_Traits,_Alloc>& __x, const _CharT* __s)
126	bool _STLP_CALL operator>=(const _CharT* __s, const basic_string<_CharT,_Traits,_Alloc>& __y)
127	bool _STLP_CALL operator>=(const basic_string<_CharT,_Traits,_Alloc>& __x, const _CharT* __s)

19. <typerinfo>

The methods supported by <typerinfo> are listed below.

List of Methods for Class typerinfo

1	~type_info()
2	__bool operator==(const type_info&)
3	__bool operator!=(const type_info&)
4	__bool before(const type_info&)
5	const char* name()

20. <utility>

The methods supported by <utility> are listed below.

List of Methods for Class utility

1	pair()
2	pair(const _T1& __a, const _T2& __b)
3	pair(const pair<_U1, _U2>& __p)
4	pair(const pair<_T1, _T2>& __o)
5	pair(__move_source<pair<_T1, _T2> > src)
6	pair<_T1, _T2 const*> make_pair(_T1 const& __x, _T2 const (&__y)[_Sz])
7	pair<_T1 const*, _T2> make_pair(_T1 const (&__x)[_Sz], _T2 const& __y)
8	pair<_T1 const*, _T2 const*> make_pair(_T1 const (&__x)[_Sz1], _T2 const (&__y)[_Sz2])
9	pair<_T1, _T2> make_pair(_T1 __x, _T2 __y)
10	bool _STLP_CALL operator==(const pair<_T1, _T2>& __x, const pair<_T1, _T2>& __y)
11	bool _STLP_CALL operator<(const pair<_T1, _T2>& __x, const pair<_T1, _T2>& __y)
12	bool _STLP_CALL operator!=(const pair<_T1, _T2>& __x, const pair<_T1, _T2>& __y)
13	bool _STLP_CALL operator>(const pair<_T1, _T2>& __x, const pair<_T1, _T2>& __y)
14	bool _STLP_CALL operator<=(const pair<_T1, _T2>& __x, const pair<_T1, _T2>& __y)
15	bool _STLP_CALL operator>=(const pair<_T1, _T2>& __x, const pair<_T1, _T2>& __y)

21. <valarray>

The methods supported by <valarray> are listed below.

List of Methods for Class valarray

1	valarray()
2	explicit valarray(size_t __n)
3	valarray(const value_type& __x, size_t __n)
4	valarray(const value_type* __p, size_t __n)
5	valarray(const valarray<_Tp>& __x)
6	valarray(const slice_array<_Tp>&)
7	valarray(const gslice_array<_Tp>&)
8	valarray(const mask_array<_Tp>&)
9	valarray(const indirect_array<_Tp>&)
10	~valarray()
11	valarray<_Tp>& operator=(const valarray<_Tp>& __x)
12	valarray<_Tp>& operator=(const value_type& __x)
13	valarray<_Tp>& operator=(const slice_array<_Tp>&)
14	valarray<_Tp>& operator=(const gslice_array<_Tp>&)
15	valarray<_Tp>& operator=(const mask_array<_Tp>&)
16	valarray<_Tp>& operator=(const indirect_array<_Tp>&)
17	value_type operator[](size_t __n)
18	value_type& operator[](size_t __n)
19	valarray<_Tp> operator[](slice) const
20	slice_array<_Tp> operator[](slice)
21	valarray<_Tp> operator[](const gslice&)
22	gslice_array<_Tp> operator[](const gslice&)
23	valarray<_Tp> operator[](const _Valarray_bool&)
24	mask_array<_Tp> operator[](const _Valarray_bool&)
25	valarray<_Tp> operator[](const _Valarray_size_t&)
26	indirect_array<_Tp> operator[](const _Valarray_size_t&)
27	size_t size()
28	valarray<_Tp> operator+()
29	valarray<_Tp> operator-()
30	valarray<_Tp> operator~()
31	_Valarray_bool operator!()
32	valarray<_Tp>& operator*= (const value_type& __x)
33	valarray<_Tp>& operator*= (const valarray<_Tp>& __x)
34	valarray<_Tp>& operator/= (const value_type& __x)
35	valarray<_Tp>& operator/= (const valarray<_Tp>& __x)
36	valarray<_Tp>& operator%= (const value_type& __x)
37	valarray<_Tp>& operator%= (const valarray<_Tp>& __x)
38	valarray<_Tp>& operator+= (const value_type& __x)
39	valarray<_Tp>& operator+= (const valarray<_Tp>& __x)
40	valarray<_Tp>& operator-= (const value_type& __x)

41	valarray<_Tp>& operator-= (const valarray<_Tp>& __x)
42	valarray<_Tp>& operator^= (const value_type& __x)
43	valarray<_Tp>& operator^= (const valarray<_Tp>& __x)
44	valarray<_Tp>& operator&= (const value_type& __x)
45	valarray<_Tp>& operator&= (const valarray<_Tp>& __x)
46	valarray<_Tp>& operator = (const value_type& __x)
47	valarray<_Tp>& operator = (const valarray<_Tp>& __x)
48	valarray<_Tp>& operator<=<= (const value_type& __x)
49	valarray<_Tp>& operator<=<= (const valarray<_Tp>& __x)
50	valarray<_Tp>& operator>=>= (const value_type& __x)
51	valarray<_Tp>& operator>=>= (const valarray<_Tp>& __x)
52	value_type sum()
53	value_type (min) ()
54	value_type (max) ()
55	valarray<_Tp> shift(int __n)
56	valarray<_Tp> cshift(int __n)
57	valarray<_Tp> apply(value_type __f(value_type))
58	valarray<_Tp> apply(value_type __f(const value_type&))
59	void resize(size_t __n, value_type __x = value_type())
60	valarray<_Tp> _STLP_CALL operator*(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
61	valarray<_Tp> _STLP_CALL operator/(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
62	valarray<_Tp> _STLP_CALL operator%(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
63	valarray<_Tp> _STLP_CALL operator+(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
64	valarray<_Tp> _STLP_CALL operator-(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
65	valarray<_Tp> _STLP_CALL operator^(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
66	valarray<_Tp> _STLP_CALL operator&(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
67	valarray<_Tp> _STLP_CALL operator (const valarray<_Tp>& __x, const valarray<_Tp>& __y)
68	valarray<_Tp> _STLP_CALL operator<<(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
69	valarray<_Tp> _STLP_CALL operator>>(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
70	valarray<_Tp> _STLP_CALL operator*(const valarray<_Tp>& __x, const _Tp& __c)
71	valarray<_Tp> _STLP_CALL operator*(const _Tp& __c, const valarray<_Tp>& __x)
72	valarray<_Tp> _STLP_CALL operator/(const valarray<_Tp>& __x, const _Tp& __c)
73	valarray<_Tp> _STLP_CALL operator/(const _Tp& __c, const valarray<_Tp>& __x)
74	valarray<_Tp> _STLP_CALL operator%(const valarray<_Tp>& __x, const _Tp& __c)
75	valarray<_Tp> _STLP_CALL operator%(const _Tp& __c, const valarray<_Tp>& __x)
76	valarray<_Tp> _STLP_CALL operator+(const valarray<_Tp>& __x, const _Tp& __c)
77	valarray<_Tp> _STLP_CALL operator+(const _Tp& __c, const valarray<_Tp>& __x)
78	valarray<_Tp> _STLP_CALL operator-(const valarray<_Tp>& __x, const _Tp& __c)
79	valarray<_Tp> _STLP_CALL operator-(const _Tp& __c, const valarray<_Tp>& __x)
80	valarray<_Tp> _STLP_CALL operator^(const valarray<_Tp>& __x, const _Tp& __c)
81	valarray<_Tp> _STLP_CALL operator^(const _Tp& __c, const valarray<_Tp>& __x)
82	valarray<_Tp> _STLP_CALL operator&(const valarray<_Tp>& __x, const _Tp& __c)
83	valarray<_Tp> _STLP_CALL operator&(const _Tp& __c, const valarray<_Tp>& __x)
84	valarray<_Tp> _STLP_CALL operator (const valarray<_Tp>& __x, const _Tp& __c)
85	valarray<_Tp> _STLP_CALL operator (const _Tp& __c, const valarray<_Tp>& __x)
86	valarray<_Tp> _STLP_CALL operator<<(const valarray<_Tp>& __x, const _Tp& __c)
87	valarray<_Tp> _STLP_CALL operator<<(const _Tp& __c, const valarray<_Tp>& __x)
88	valarray<_Tp> _STLP_CALL operator>>(const valarray<_Tp>& __x, const _Tp& __c)
89	valarray<_Tp> _STLP_CALL operator>>(const _Tp& __c, const valarray<_Tp>& __x)

90	_Valarray_bool_STLP_CALL operator==(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
91	_Valarray_bool_STLP_CALL operator>(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
92	_Valarray_bool_STLP_CALL operator!=(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
93	_Valarray_bool_STLP_CALL operator>(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
94	_Valarray_bool_STLP_CALL operator<=(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
95	_Valarray_bool_STLP_CALL operator>=(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
96	_Valarray_bool_STLP_CALL operator&&(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
97	_Valarray_bool_STLP_CALL operator (const valarray<_Tp>& __x, const valarray<_Tp>& __y)
98	_Valarray_bool_STLP_CALL operator==(const valarray<_Tp>& __x, const _Tp& __c)
99	_Valarray_bool_STLP_CALL operator==(const _Tp& __c, const valarray<_Tp>& __x)
100	_Valarray_bool_STLP_CALL operator!=(const valarray<_Tp>& __x, const _Tp& __c)
101	_Valarray_bool_STLP_CALL operator!=(const _Tp& __c, const valarray<_Tp>& __x)
102	_Valarray_bool_STLP_CALL operator<(const valarray<_Tp>& __x, const _Tp& __c)
103	_Valarray_bool_STLP_CALL operator<(const _Tp& __c, const valarray<_Tp>& __x)
104	_Valarray_bool_STLP_CALL operator>(const valarray<_Tp>& __x, const _Tp& __c)
105	_Valarray_bool_STLP_CALL operator>(const _Tp& __c, const valarray<_Tp>& __x)
106	_Valarray_bool_STLP_CALL operator<=(const valarray<_Tp>& __x, const _Tp& __c)
107	_Valarray_bool_STLP_CALL operator<=(const _Tp& __c, const valarray<_Tp>& __x)
108	_Valarray_bool_STLP_CALL operator>=(const valarray<_Tp>& __x, const _Tp& __c)
109	_Valarray_bool_STLP_CALL operator>=(const _Tp& __c, const valarray<_Tp>& __x)
110	_Valarray_bool_STLP_CALL operator&&(const valarray<_Tp>& __x, const _Tp& __c)
111	_Valarray_bool_STLP_CALL operator&&(const _Tp& __c, const valarray<_Tp>& __x)
112	_Valarray_bool_STLP_CALL operator (const valarray<_Tp>& __x, const _Tp& __c)
113	_Valarray_bool_STLP_CALL operator (const _Tp& __c, const valarray<_Tp>& __x)
114	valarray<_Tp> abs(const valarray<_Tp>& __x)
115	valarray<_Tp> acos(const valarray<_Tp>& __x)
116	valarray<_Tp> asin(const valarray<_Tp>& __x)
117	valarray<_Tp> atan(const valarray<_Tp>& __x)
118	valarray<_Tp> atan2(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
119	valarray<_Tp> atan2(const valarray<_Tp>& __x, const _Tp& __c)
120	valarray<_Tp> atan2(const _Tp& __c, const valarray<_Tp>& __x)
121	valarray<_Tp> cos(const valarray<_Tp>& __x)
122	valarray<_Tp> cosh(const valarray<_Tp>& __x)
123	valarray<_Tp> exp(const valarray<_Tp>& __x)
124	valarray<_Tp> log(const valarray<_Tp>& __x)
125	valarray<_Tp> log10(const valarray<_Tp>& __x)
126	valarray<_Tp> pow(const valarray<_Tp>& __x, const valarray<_Tp>& __y)
127	valarray<_Tp> pow(const valarray<_Tp>& __x, const _Tp& __c)
128	valarray<_Tp> pow(const _Tp& __c, const valarray<_Tp>& __x)
129	valarray<_Tp> sin(const valarray<_Tp>& __x)
130	valarray<_Tp> sinh(const valarray<_Tp>& __x)
131	valarray<_Tp> sqrt(const valarray<_Tp>& __x)
132	valarray<_Tp> tan(const valarray<_Tp>& __x)
133	valarray<_Tp> tanh(const valarray<_Tp>& __x)
134	slice()
135	slice(size_t __start, size_t __length, size_t __stride)
136	size_t start()
137	size_t size()
138	size_t stride()

139	void operator=(const valarray<value_type>& __x)
140	void operator=(const value_type& __c)
141	void operator*=(const valarray<value_type>& __x)
142	void operator/=(const valarray<value_type>& __x)
143	void operator%=(const valarray<value_type>& __x)
144	void operator+=(const valarray<value_type>& __x)
145	void operator-=(const valarray<value_type>& __x)
146	void operator^=(const valarray<value_type>& __x)
147	void operator&=(const valarray<value_type>& __x)
148	void operator =(const valarray<value_type>& __x)
149	void operator<<=(const valarray<value_type>& __x)
150	void operator>>=(const valarray<value_type>& __x)
151	slice_array(const slice_array & __x)
152	~slice_array()
153	gslice()
154	gslice(size_t __start, const _Valarray_size_t& __lengths, const _Valarray_size_t& __strides)
155	start()
156	_Valarray_size_t size()
157	_Valarray_size_t stride()
158	bool _M_empty()
159	size_t _M_size()
160	void operator= (const valarray<value_type>& __x)
161	void operator= (const value_type& __c)
162	void operator*= (const valarray<value_type>& __x)
163	void operator/= (const valarray<value_type>& __x)
164	void operator%= (const valarray<value_type>& __x)
165	void operator+= (const valarray<value_type>& __x)
166	void operator-= (const valarray<value_type>& __x)
167	void operator^= (const valarray<value_type>& __x)
168	void operator&= (const valarray<value_type>& __x)
169	void operator = (const valarray<value_type>& __x)
170	void operator<<= (const valarray<value_type>& __x)
171	void operator>>= (const valarray<value_type>& __x)
172	gslice_array(const gslice_array& __x)
173	~gslice_array()
174	void operator=(const valarray<value_type>& __x)
175	void operator=(const value_type& __c)
176	void operator*=(const valarray<value_type>& __x)
177	void operator/=(const valarray<value_type>& __x)
178	void operator%=(const valarray<value_type>& __x)
179	void operator+=(const valarray<value_type>& __x)
180	void operator-=(const valarray<value_type>& __x)
181	void operator^=(const valarray<value_type>& __x)
182	void operator&=(const valarray<value_type>& __x)
183	void operator =(const valarray<value_type>& __x)
184	void operator<<=(const valarray<value_type>& __x)
185	void operator>>=(const valarray<value_type>& __x)
186	mask_array(const mask_array& __x)
187	~mask_array()

188	void operator=(const valarray<value_type>& __x)
189	void operator=(const value_type& __c)
190	void operator*=(const valarray<value_type>& __x)
191	void operator/=(const valarray<value_type>& __x)
192	void operator%=(const valarray<value_type>& __x)
193	void operator+=(const valarray<value_type>& __x)
194	void operator-=(const valarray<value_type>& __x)
195	void operator^=(const valarray<value_type>& __x)
196	void operator&=(const valarray<value_type>& __x)
197	void operator =(const valarray<value_type>& __x)
198	void operator<<=(const valarray<value_type>& __x)
199	void operator>>=(const valarray<value_type>& __x)
200	indirect_array(const indirect_array& __x)
201	~indirect_array()

22. <vector>

The methods supported by <vector> are listed below.

List of Methods for Class vector

1	allocator_type get_allocator()
2	iterator begin()
3	const_iterator begin()
4	iterator end()
5	const_iterator end()
6	reverse_iterator rbegin()
7	const_reverse_iterator rbegin()
8	reverse_iterator rend()
9	const_reverse_iterator rend()
10	size_type size()
11	size_type max_size()
12	size_type capacity()
13	bool empty()
14	reference front()
15	const_reference front()
16	reference back()
17	const_reference back()
18	reference at(size_type __n)
19	const_reference at(size_type __n)
20	vector(const allocator_type& __a = allocator_type())
21	vector(size_type __n)
22	vector(const _Self& __x)
23	vector(__move_source<_Self> src)
24	vector(_InputIterator __first, _InputIterator __last, const allocator_type& __a = allocator_type())
25	void reserve(size_type __n)
26	void assign(size_type __n, const Tp& __val)
27	void _M_fill_assign(size_type __n, const Tp& __val)
28	void push_back(const Tp& __x = Tp())
29	iterator insert(iterator __pos, const Tp& __x = Tp())
30	void insert(iterator __pos, _InputIterator __first, _InputIterator __last)
31	void insert (iterator __pos, size_type __n, const Tp& __x)
32	void swap(_Self& __x)
33	void pop_back()
34	iterator erase(iterator __pos)
35	iterator erase(iterator __first, iterator __last)
36	void resize(size_type __new_size, const Tp& __x = _STLP_DEFAULT_CONSTRUCTED(_Tp))
37	void clear()
38	reference operator[] (size_type __n)
39	_Self& operator= (const _Self& __x)
40	bool _STLP_CALL operator== (const vector<Tp>& __x, const vector<Tp>& __y)
41	bool _STLP_CALL operator< (const vector<Tp>& __x, const vector<Tp>& __y)
42	bool _STLP_CALL operator!= (const vector<Tp>& __x, const vector<Tp>& __y)
43	bool _STLP_CALL operator<= (const vector<Tp>& __x, const vector<Tp>& __y)
44	bool _STLP_CALL operator> (const vector<Tp>& __x, const vector<Tp>& __y)
45	bool _STLP_CALL operator>= (const vector<Tp>& __x, const vector<Tp>& __y)

Website and Support <website and support,ws>

Renesas Electronics Website

<http://www.renesas.com/>

Inquiries

<http://www.renesas.com/inquiry>

All trademarks and registered trademarks are the property of their respective owners.

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins

Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

- The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

The state of the product is undefined at the moment when power is supplied.

- The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.

In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.

In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

Access to reserved addresses is prohibited.

- The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

After applying a reset, only release the reset line after the operating clock signal has become stable.

When switching the clock signal during program execution, wait until the target clock signal has stabilized.

- When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

Before changing from one product to another, i.e. to one with a different type number, confirm that the change will not lead to problems.

- The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
"Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
"High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
"Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.



SALES OFFICES

Renesas Electronics Corporation

<http://www.renesas.com>

Refer to "<http://www.renesas.com/>" for the latest and detailed information.

Renesas Electronics America Inc.

2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited

1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited

Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K.
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH

Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.

7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.

Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited

Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852-2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.

7F, No. 363 Fu Shing North Road Taipei, Taiwan, R.O.C.
Tel: +886-2-8175-9600, Fax: +886-2-8175-9670

Renesas Electronics Singapore Pte. Ltd.

1 HarbourFront Avenue, #06-10, Keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.

Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.

11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141