第34章 STL算法

由标准模板库定义的算法在这里描述，这些算法通过迭代器对容器进行操作。所有的算法都是模板函数，下面是由算法使用的通用类型名称的描述。

通用名称	表示
BiIter	双向迭代器
ForIter	前向迭代器
InIter	输入迭代器
OutIter	输出迭代器
RandIter	随机访问迭代器
T	某种类型的数据
Size	某种类型的整数
Func	某种类型的函数
Generator	生成对象的函数
BinPred	二元谓词
UnPred	一元谓词
Comp	比较函数

34.1 adjacent_find

```
template <class ForIter>
  ForIter adjacent_find(ForIter start, ForIter end);
template <class ForIter, class BinPred>
  ForIter adjacent_find(ForIter start, ForIter end, BinPred pfn);
```

adjacent_find( )算法查找在一个序列内临近的匹配元素（这个序列由start和end指定）并返回到第一个元素的迭代器。如果没有找到临近的匹配对，返回end。第一种形式查找对应的元素。第二种形式让你指定决定匹配元素的方法。

34.2 binary_searcb

```
template <class ForIter, class T>
  bool binary_search(ForIter start, ForIter end, const T &val);
template <class ForIter, class T, class Comp>
  bool binary_search(ForIter start, ForIter end, const T &val,
                     Comp cmpfn);
```

对于由val指定的值，binary_search( )算法对在start处开始、在end处结束的有序序列执行二分查找。如果找到val，它返回真。否则，返回假。第一种形式比较指定序列中元素是否相等。第二种形式允许你指定自己的比较函数。

34.3 copy

```
template <class InIter, class OutIter>
  OutIter copy(InIter start, InIter end, OutIter result);
```

copy( )算法复制从 start 开始、在 end 结束的一个序列，并把结果放到 result 所指的序列中。它返回一个到所得序列末尾的迭代器。要复制的范围不应该与 result 重叠。

34.4 copy_backward

```
template <class BiIter1, class BiIter2>
  BiIter2 copy_backward(BiIter1 start, BiIter1 end, BiIter2 result);
```

copy_backward( )算法与 copy( )算法相同，只是它从序列的末尾向前移动元素。

34.5 count

```
template <class InIter, class T>
  ptrdiff_t count(InIter start, InIter end, const T &val);
```

count( )算法返回序列中从 start 开始、在 end 结束且匹配 val 的元素数。

34.6 count_if

```
template <class InIter, class UnPred>
  ptrdiff_t count(InIter start, InIter end, UnPred pfn);
```

count_if( )算法返回序列中从 start 开始、在 end 结束且其一元谓词 pfn 返回真的元素数。类型 ptrdiff_t 被定义为某种形式的整数。

34.7 equal

```
template <class InIter1, class InIter2>
  bool equal(InIter1 start1, InIter1 end1, InIter2 start2);
template <class InIter1, class InIter2, class BinPred>
  bool equal(InIter1 start1, InIter1 end1, InIter2 start2,
             BinPred pfn);
```

equal( )算法确定两个范围是否相同。由 start1 和 end1 确定的范围依据 start2 所指的序列进行测试。如果范围相同，那么返回真；否则，返回假。

第二种形式允许你指定一个二分查找，该二分查找决定两个元素何时相等。

34.8 equal_range

```
template <class ForIter, class T>
  pair<ForIter, ForIter> equal_range(ForIter start, ForIter end,
                                     const T &val);
template <class ForIter, class T, class Comp>
  pair<ForIter, ForIter> equal_range(ForIter start, ForIter end,
                                     const T &val, Comp cmpfn);
```

equal_range( )算法返回一个范围，在这个范围中，可以把一个元素插入到一个序列中而不破坏序列的顺序。查找这样一个范围所在的区域由 start 和 end 指定。值在 val 中传递。要指定自己的查找规则，应指定比较函数 cmpfn。

模板类 pair 是一个实用工具类，可以包含在它的 first 和 second 成员中的一对对象。

34.9 fill 和 fill_n

```
template <class ForIter, class T>
  void fill(ForIter start, ForIter end, const T &val);
template <class OutIter, class Size, class T>
  void fill_n(OutIter start, Size num, const T &val);
```

fill( )和 fill_n( )算法用 val 指定的值填充一个范围。对于 fill( )，范围由 start 和 end 指定。对于 fill_n( )，范围由 start 开始，共包含 num 个元素。

34.10 find

```
template <class InIter, class T>
  InIter find(InIter start, InIter end, const T &val);
```

对于由 val 指定的值，find( )算法查找从 start 开始到 end 结束的范围。它返回指向该元素第一次出现处的迭代器或者如果该值不在序列中，则返回一个指向 end 的迭代器。

34.11 find_end

```
template <class ForIter1, class ForIter2>
  ForIter1 find_end(ForIter1 start1, ForIter1 end1,
                    ForIter2 start2, ForIter2 end2);
template <class ForIter1, class ForIter2, class BinPred>
  ForIter1 find_end(ForIter1 start1, ForIter1 end1,
                    ForIter2 start2, ForIter2 end2, BinPred pfn);
```

find_end( )算法查找在 start1 和 end1 定义的范围内由 start2 和 end2 所定义的最后一个序列。如果找到这个序列，返回指向该序列中第一个元素的迭代器。否则，返回迭代器 end1。

第二种形式允许你指定一个二元谓词，该谓词决定何时元素匹配。

34.12 find_first_of

```
template <class ForIter1, class ForIter2>
  ForIter1 find_first_of(ForIter1 start1, ForIter1 end1,
                         ForIter2 start2, ForIter2 end2);
template <class ForIter1, class ForIter2, class BinPred>
  ForIter1 find_first_of(ForIter1 start1, ForIter1 end1,
                         ForIter2 start2, ForIter2 end2,
                         BinPred pfn);
```

find_first_of( )算法查找在 start1 和 end1 所定义的序列内匹配 start2 和 end2 定义的序列内一个元素的第一个元素。如果没有找到匹配的元素，返回 end1 迭代器。

第二种形式允许你指定一个二元谓词，该谓词决定何时元素匹配。

34.13　find_if

```
template <class InIter, class UnPred>
  InIter find_if(InIter start, InIter end, UnPred pfn);
```

find_if( ) 算法查找由 start 和 end 指定的范围，以找到某一元素，对于该元素，一元谓词 pfn 返回真。它返回指向该元素第一次出现处的迭代器。如果值不在这个序列中，则返回指向 end 的一个迭代器。

34.14　for_each

```
teplate<class InIter, class Func>
  Func for_each(InIter start, InIter end, Func fn);
```

for_each( )算法应用函数 fn 到由 start 和 end 指定的元素范围。它返回 fn。

34.15　generate 和 generate_n

```
template <class ForIter, class Generator>
  void generate(ForIter start, ForIter end, Generator fngen);
template <class ForIter, class Size, class Generator>
  void generate_n(OutIter start, Size num, Generator fngen);
```

算法 generate( )和 generate_n( )给某一范围的元素赋予由一个生成器函数返回的值。对于 generate( )，被赋值的范围由 start 和 end 指定。对于 generate_n( )，范围从 start 开始，共包含 num 个元素。生成器函数在 fngen 中传递，它不带参数。

34.16　includes

```
template <class InIter1, class InIter2>
  bool includes(InIter1 start1, InIter1 end1,
                InIter2 start2, InIter2 end2);
template <class InIter1, class InIter2, class Comp>
  bool includes(InIter1 start1, InIter1 end1,
                InIter2 start2, InIter2 end2, Comp cmpfn);
```

includes( ) 算法决定由 start1 和 end1 定义的序列是否包括由 start2 和 end2 定义的序列内的所有元素。如果所有元素都能找到，它返回真。否则，返回假。第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.17　inplace_merge

```
template <class BiIter>
  void inplace_merge(BiIter start, BiIter mid, BiIter end);
template <class BiIter, class Comp>
  void inplace_merge(BiIter start, BiIter mid, BiIter end, Comp cmpfn);
```

在一个序列内，inplace_merge( )算法把由 start 和 mid 所定义的范围和由 mid 和 end 所定义的范围合并。两个范围必须按升序存储。执行后，所得的结果序列按升序排序。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.18 iter_swap

```
template <class ForIter1, class ForIter2>
  void iter_swap(ForIter1 i, ForIter2 j)
```

iter_swap( )算法交换由它的两个迭代器参数所指向的值。

34.19 lexicographical_compare

```
template <class InIter1, class InIter2>
  bool lexicographical_compare(InIter1 start1, InIter1 end1,
                               InIter2 start2, InIter2 end2);
template <class InIter1, class InIter2, class Comp>
  bool lexicographical_compare(InIter1 start1, InIter1 end1,
                               InIter2 start2, InIter2 end2,
                               Comp cmpfn);
```

lexicographical_compare( )算法按字母顺序比较由start1、end1定义的序列和由start2、end2定义的序列。如果按词汇统计法第一个序列小于第二个序列（即，如果第一个序列按字典顺序出现在第二个序列之前），它返回真。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.20 lower_bound

```
template <class ForIter, class T>
  ForIter lower_bound(ForIter start, ForIter end, const T &val);
template <class ForIter, class T, class Comp>
  ForIter lower_bound(ForIter start, ForIter end, const T &val,
                      Comp cmpfn);
```

lower_bound( )算法查找由start和end定义的序列中不小于val的第一个位置。它返回一个指向此位置的迭代器。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.21 make_heap

```
template <class RandIter>
  void make_heap(RandIter start, RandIter end);
template <class RandIter, class Comp>
  void make_heap(RandIter start, RandIter end, Comp cmpfn);
```

make_heap( )算法从start和end定义的序列中构造一个堆。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.22 max

```
template <class T>
```

```
  const T &max(const T &i, const T &j);
template <class T, class Comp>
  const T &max(const T &i, const T &j, Comp cmpfn);
```

max( )算法返回两个值中的最大值。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.23 max_element

```
template <class ForIter>
  ForIter max_element(ForIter start, ForIter last);
template <class ForIter, class Comp>
  ForIter max_element(ForIter start, ForIter last, Comp cmpfn);
```

max_element( ) 算法返回一个指向由 start 和 end 定义的范围内最大元素的迭代器。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.24 merge

```
template <class InIter1, class InIter2, class OutIter>
  OutIter merge(InIter1 start1, InIter1 end1,
                InIter2 start2, InIter2 end2,
                OutIter result);
template <class InIter1, class InIter2, class OutIter, class Comp>
  OutIter merge(InIter1 start1, InIter1 end1,
                InIter2 start2, InIter2 end2,
                OutIter result, Comp cmpfn);
```

merge( ) 算法合并两个有序序列，把结果放到第三个序列中。要合并的序列由 start1、end1 和 start2、end2 定义。结果被放到由 result 所指向的序列中。一个指向结果序列末尾的迭代器被返回。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.25 min

```
template <class T>
  const T &min(const T &i, const T &j);
template <class T, class Comp>
  const T &min(const T &i, const T &j, Comp cmpfn);
```

min( ) 算法返回两个值中的最小值。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.26 min_element

```
template <class ForIter>
  ForIter min_element(ForIter start, ForIter last);
template <class ForIter, class Comp>
  ForIter min_element(ForIter start, ForIter last, Comp cmpfn);
```

min_element( )算法返回一个指向 start 和 last 所定义范围内最小元素的迭代器。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.27 mismatch

```
template <class InIter1, class InIter2>
  pair<InIter1, InIter2> mismatch(InIter1 start1, InIter1 end1,
                                  InIter2 start2);
template <class InIter1, class InIter2, class BinPred>
  pair<InIter1, InIter2> mismatch(InIter1 start1, InIter1 end1,
                                  InIter2 start2, BinPred pfn);
```

mismatch( )算法查找在两个序列中元素之间第一个不匹配的情况。返回指向两个元素的迭代器。如果没有找到不匹配的元素，返回指向序列末尾的迭代器。

第二种形式允许你指定一个二元谓词，该谓词决定何时一个元素等于另一个元素。

pair 模板类包含两个数据成员，称为 first 和 second ，用于容纳值对。

34.28 next_permutation

```
template <class BiIter>
  bool next_permutation(BiIter start, BiIter end);
template <class BiIter, class Comp>
  bool next_permutation(BiIter start, BiIter end, Comp cmfn);
```

next_permutation( )算法构造一个序列的下一种排列。依据一个排好序的序列生成这个排列从低到高表示第一个排列。如果下一个排列不存在，next_permutation( )排序这个序列作为它的第一个排列并返回假。否则，它返回真。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.29 nth_element

```
template <class RandIter>
  void nth_element(RandIter start, RandIter element, RandIter end);
template <class RandIter, class Comp>
  void nth_element(RandIter start, RandIter element,
                   RandIter end, Comp cmpfn);
```

nth_element( ) 算法安排由 start 和 end 指定的序列，以便所有小于 element 的元素位于那个元素（即 element ）之前，所有大于 element 的元素位于其后。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素大于另一个元素。

34.30 partial_sort

```
template <class RandIter>
  void partial_sort(RandIter start, RandIter mid, RandIter end);
template <class RandIter, class Comp>
  void partial_sort(RandIter start, RandIter mid,
                    RandIter end, Comp cmpfn);
```

partial_sort( )算法排序由 start 和 end 指定的范围。然而，在执行后，只有在 start 到 mid 之间的元素才是排好序的。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.31 partial_sort_copy

```
template <class InIter, class RandIter>
  RandIter partial_sort_copy(InIter start, InIter end,
                             RandIter res_start, RandIter res_end);
template <class InIter, class RandIter, class Comp>
  RandIter partial_sort_copy(InIter start, InIter end,
                             RandIter res_start, RandIter res_end,
                             Comp cmpfn);
```

partial_sort_copy( )算法排序由 start 和 end 指定的范围，然后把合适数量的元素填充到由 res_start 和 res_end 定义的结果序列中。它返回一个迭代器，该迭代器指向复制到结果序列中的最后一个元素后面的一个元素。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.32 partition

```
template <class BiIter, class UnPred>
  BiIter partition(BiIter start, BiIter end, UnPred pfn);
```

partition( )算法安排由 start 和 end 定义的序列，以便使所有满足特定条件的元素（该特定条件是：对于这些元素，由 pfn 指定的谓词返回真）位于另一些元素之前（对于这另一些元素，谓词返回假）。它返回一个迭代器，该迭代器指向其谓词返回假的元素的开始处。

34.33 pop_heap

```
template <class RandIter>
  void pop_heap(RandIter start, RandIter end);
template <class RandIter, class Comp>
  void pop_heap(RandIter start, RandIter end, Comp cmpfn);
```

pop_heap( ) 算法把 first 和 last-1 元素相交换，然后重新构建堆。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.34 prev_permutation

```
template <class BiIter>
  bool prev_permutation(BiIter start, BiIter end);
template <class BiIter, class Comp>
  bool prev_permutation(BiIter start, BiIter end, Comp cmpfn);
```

prev_permutation( ) 算法构建一个序列先前的（previous）排列。这个排列依据一个排好序的序列生成：从低到高表示第一个排列。如果下一个排列不存在，prev_permutation( )排序该序列作为它的最终排列并返回假。否则，它返回真。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.35 push_heap

```
template <class RandIter>
  void push_heap(RandIter start, RandIter end);
template <class RandIter, class Comp>
  void push_heap(RandIter start, RandIter end, Comp cmpfn);
```

push_heap( ) 算法把一个元素压入堆中。由 start 和 end 指定的范围被假定为代表一个有效的堆。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.36 random_shuffle

```
template <class RandIter>
  void random_shuffle(RandIter start, RandIter end);
template <class RandIter, class Generator>
  void random_shuffle(RandIter start, RandIter end, Generator rand_gen);
```

random_shuffle( )算法随机化由 start 和 end 定义的序列。

第二种形式指定一个自定义随机数生成器。这个函数必须具有下面的一般形式：

```
rand_gen(num);
```

它必须返回一个位于 0 和 num 之间的随机数。

34.37 remove, remove_if, remove_copy 和 remove_copy_if

```
template <class ForIter, class T>
  ForIter remove(ForIter start, ForIter end, const T &val);
template <class ForIter, class UnPred>
  ForIter remove_if(ForIter start, ForIter end, UnPred pfn);
template <class InIter, class OutIter, class T>
  OutIter remove_copy(InIter start, InIter end,
                      OutIter result, const T &val);
template <class InIter, class OutIter, class UnPred>
  OutIter remove_copy_if(InIter start, InIter end,
                         OutIter result, UnPred pfn);
```

remove( )算法从指定的范围中删去等于 val 的元素。它返回一个指向所剩余元素末尾的迭代器。

remove_if( )算法从指定范围（在此范围谓词 pfn 为真）中删除元素。它返回一个指向所剩余元素末尾的迭代器。

remove_copy( ) 算法从指定范围复制等于 val 的元素并把结果放到 result 所指向的序列中。它返回一个指向结果末尾的迭代器。

remove_copy_if( )算法从指定范围（在此范围谓词 pfn 为真）复制元素并把结果放到由 result 所指向的序列中。它返回一个指向结果末尾的迭代器。

34.38　replace, replace_copy, replace_if 和 replace_copy_if

```
template <class ForIter, class T>
  void replace(ForIter start, ForIter end,
               const T &old, const T &new);
template <class ForIter, class UnPred, class T>
  void replace_if(ForIter start, ForIter end,
                  UnPred pfn, const T &new);
template <class InIter, class OutIter, class T>
  OutIter replace_copy(InIter start, InIter end, OutIter result,
                       const T &old, const T &new);
template <class InIter, class OutIter, class UnPred, class T>
  OutIter replace_copy_if(InIter start, InIter end, OutIter result,
                          UnPred pfn, const T &new);
```

在指定的范围内，replace( )算法用值为 new 的元素取代值为 old 的元素。

在指定的范围内，replace_if( ) 算法用值为 new 的元素取代其谓词 pfn 为真的那些元素。

在指定的范围内，replace_copy( )算法把元素复制到result中。在这个过程中，它用值为 new 的元素取代值为 old 的元素。初始范围不变。返回一个指向 result 末尾的迭代器。

在指定的范围内，replace_copy_if( )算法把元素复制到 result 中。在这个过程中，它用值为 new的元素取代使其谓词pfn返回真的元素。初始范围不变。返回一个指向result末尾的迭代器。

34.39　reverse 和 reverse_copy

```
template <class BiIter>
  void reverse(BiIter start, BiIter end);
template <class BiIter, class OutIter>
  OutIter reverse_copy(BiIter start, BiIter end, OutIter result);
```

reverse( ) 算法颠倒由 start 和 end 指定的范围的次序。

reverse_copy( ) 算法按逆序复制由 start 和 end 所指定的范围，并把结果存储在 result 中。它返回一个指向 result 末尾的迭代器。

34.40　rotate 和 rotate_copy

```
template <class ForIter>
  void rotate(ForIter start, ForIter mid, ForIter end);
template <class ForIter, class OutIter>
  OutIter rotate_copy(ForIter start, ForIter mid, ForIter end,
                      OutIter result);
```

rotate( )算法左旋由 start和end指定的范围内的元素，以便mid所指的元素变成新的第一个元素。

rotate_copy( )算法复制由 start 和 end 指定的范围，并把结果存储在 result 中。在这个过程中，它左旋这些元素，以便 mid 所指的元素成为新的第一个元素。它返回一个指向 result 末尾的迭代器。

34.41 search

```
template <class ForIter1, class ForIter2>
  ForIter1 search(ForIter1 start1, ForIter1 end1,
                  ForIter2 start2, ForIter2 end2);
template <class ForIter1, class ForIter2, class BinPred>
  ForIter1 search(ForIter1 start1, ForIter1 end1,
                  ForIter2 start2, ForIter2 end2, BinPred pfn);
```

search( )算法查找一个序列中的子序列，所查找的序列由start1和end1定义，所查找的子序列由start2和end2指定。如果找到了子序列，返回指向其开始处的迭代器。否则，返回end1。

第二种形式允许你指定一个二元谓词，该谓词决定何时一个元素等于另一个元素。

34.42 search_n

```
template <class ForIter, class Size, class T>
  ForIter search_n(ForIter start, ForIter end,
                   Size num, const T &val);
template <class ForIter, class Size, class T, class BinPred>
  ForIter search_n(ForIter start, ForIter end,
                   Size num, const T &val, BinPred pfn);
```

search_n( )算法查找在一个序列内等于val的num个元素组成的序列，所查找的序列由start1和end1所定义。如果找到这个子序列，返回指向它的开始处的迭代器。否则，返回end。

第二种形式允许你指定一个二元谓词，该谓词决定何时一个元素等于另一个元素。

34.43 set_difference

```
template <class InIter1, class InIter2, class OutIter>
  OutIter set_difference(InIter1 start1, InIter1 end1,
            InIter2 start2, InIter2 end2, OutIter result);
template <class InIter1, class InIter2, class OutIter, class Comp>
  OutIter set_difference(InIter1 start1, InIter1 end1,
            InIter2 start2, InIter2 end2,
            OutIter result, Comp cmpfn);
```

set_difference( )算法生成一个序列，该序列包含由start1、end1和start2、end2定义的两个有序集合之间的差。即，由start2、end2定义的集合被从由start1、end1定义的集合中减去。结果被排序并放到result中。它返回一个指向结果集末尾的迭代器。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.44 set_intersection

```
template <class InIter1, class InIter2, class OutIter>
  OutIter set_intersection(InIter1 start1, InIter1 end1,
            InIter2 start2, InIter2 end2, OutIter result);
template <class InIter1, class InIter2, class OutIter, class Comp>
  OutIter set_intersection(InIter1 start1, InIter1 end1,
            InIter2 start2, InIter2 end2,
```

```
            OutIter result, Comp cmpfn);
```

set_intersection( )算法生成一个序列，该序列包含由 start1、end1 和 start2、end2 定义的两个有序集合的交。这些是两个集合中都存在的元素。结果被排序并放到 result 中。它返回一个指向结果集合末尾的迭代器。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.45　set_symmetric_difference

```
template <class InIter1, class InIter2, class OutIter>
  OutIter set_symmetric_difference(InIter1 start1, InIter1 end1,
            InIter2 start2, InIter2 end2, OutIter result);
template <class InIter1, class InIter2, class OutIter, class Comp>
  OutIter set_symmetric_difference(InIter1 start1, InIter1 end1,
            InIter2 start2, InIter2 end2, OutIter result,
            Comp cmpfn);
```

set_symmetric_difference( )算法生成一个序列，这个序列包含由 start1、end1 和 start2、end2 所定义的两个有序集之间的差（symmetric difference），即，结果集仅包含不同时存在于两个集合中的元素。结果被排序并放到 result 中。它返回一个指向结果集末尾的迭代器。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.46　set_union

```
template <class InIter1, class InIter2, class OutIter>
  OutIter set_union(InIter1 start1, InIter1 end1,
            InIter2 start2, InIter2 end2, OutIter result);
template <class InIter1, class InIter2, class OutIter, class Comp>
  OutIter set_union(InIter1 start1, InIter1 end1,
            InIter2 start2, InIter2 end2, OutIter result,
            Comp cmpfn);
```

set_union( )算法生成一个包含由 start1、end1 和 start2,end2 定义的两个有序集的并的序列。因此，结果集包含同时在两个集合中的元素。结果被排序并放到 result 中。它返回一个指向结果集末尾的迭代器。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.47　sort

```
template <class RandIter>
  void sort(RandIter start, RandIter end);
template <class RandIter, classComp>
  void sort(RandIter start, RandIter end, Comp cmpfn);
```

sort( )算法对由 start 和 end 指定的范围进行排序。第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.48 sort_heap

```
template <class RandIter>
  void sort_heap(RandIter start, RandIter end);
template <class RandIter, class Comp>
  void sort_heap(RandIter start, RandIter end, Comp cmpfn);
```

sort_heap( ) 算法对在由 start 和 end 指定范围内的一个堆进行排序。

第二种形式允许你指定一个比较函数，该函数决定何时一个元素小于另一个元素。

34.49 stable_partition

```
template <class BiIter, class UnPred>
  BiIter stable_partition(BiIter start, BiIter end, UnPred pfn);
```

stable_partition( )算法安排由 start 和 end 定义的序列，以便使由 pfn 指定的谓词返回真的所有元素会在其谓词返回假的那些元素之前出现。分区是稳定的，这意味着序列的相对排序被保存。它返回一个指向其谓词为假的元素的开始处的迭代器。

34.50 stable_sort

```
template <class RandIter>
  void stable_sort(RandIter start, RandIter end);
template <class RandIter, class Comp>
  void stable_sort(RandIter start, RandIter end, Comp cmpfn);
```

sort( )算法对由 start 和 end 所指定的序列进行排序。这种排序是稳定的，这意味着相等的元素不会被重新安排。

第二种形式允许你指定一个比较函数，这个比较函数决定何时一个元素小于另一个元素。

34.51 swap

```
template <class T>
  void swap(T &i, T &j);
```

swap( )算法把 i 和 j 所引用的值进行交换。

34.52 swap_ranges

```
template <class ForIter1, class ForIter2>
  ForIter2 swap_ranges(ForIter1 start1, ForIter1 end1,
                       ForIter2 start2);
```

swap_ranges( )算法把由 start1 和 end1 指定的范围内的元素和以 start2 开始的序列中的元素相交换。它返回一个指向由 start2 所指定的序列末尾的迭代器。

34.53 transform

```
template <class InIter, class OutIter, class Func>
```

```
  OutIter transform(InIter start, InIter end,
                    OutIter result, Func unaryfunc);
template <class InIter1, class InIter2, class OutIter, class Func>
  OutIter transform(InIter1 start1, InIter1 end1,
                    InIter2 start2, OutIter result,
                    Func binaryfunc);
```

transform( )算法在某一元素范围应用一个函数并在result中存储结果。在第一种形式中，范围由start和end指定，所用的函数由unaryfunc指定。这个函数收到一个元素的值作为它的参数，并且它必须返回它的转换形式。

在第二种形式中，使用一个二元运算符函数来应用转换，该二元运算符函数从要被转换的序列中接收一个元素的值作为它的第一个参数，从第二个序列中接收一个元素作为第二个参数。

这两种形式都返回一个指向结果序列末尾的迭代器。

34.54 unique和unique_copy

```
template <class ForIter>
  ForIter unique(ForIter start, ForIter end);
template <class ForIter, class BinPred>
  ForIter unique(ForIter start, ForIter end, BinPred pfn);
template <class ForIter, class OutIter>
  OutIter unique_copy(ForIter start, ForIter end, OutIter result);
template <class ForIter, class OutIter, class BinPred>
  OutIter unique_copy(ForIter start, ForIter end, OutIter result,
                      BinPred pfn);
```

unique( )算法从特定范围删除重复的元素。第二种形式允许你指定一个二元谓词，该谓词决定何时一个元素等于另一个元素。unique( )返回一个指向范围末尾的迭代器。

unique_copy( )算法复制由start1和end1所指定的范围，在这个过程中删除重复的元素，结果被放到result中。第二种形式允许你指定一个二元谓词，它决定何时一个元素等于另一个元素。unique_copy( )返回一个指向范围末尾的迭代器。

34.55 upper_bouud

```
template <class ForIter, class T>
  ForIter upper_bound(ForIter start, ForIter end, const T &val);
template <class ForIter, class T, class Comp>
  ForIter upper_bound(ForIter start, ForIter end, const T &val,
                      Comp cmpfn);
```

upper_bound( )算法找到start和end定义的序列中不大于val的最后一处，它返回指向该处的迭代器。第二种形式允许你指定一个决定何时一个元素比另一个小的比较函数。