参考：<https://zh.cppreference.com/w/%E9%A6%96%E9%A1%B5>

主要基于C++ 98

算法库提供大量用途的函数（例如查找、排序、计数、操作），它们在元素范围上操作。注意范围定义为 [first, last) ，其中 last 指代要查询或修改的最后元素的*后一个*元素。

头文件<algorithm>

# 不修改序列的操作

## for\_each

template< class InputIt, class UnaryFunction >  
UnaryFunction for\_each( InputIt first, InputIt last, UnaryFunction f );

* 按顺序应用给定的函数对象 f 到解引用范围 [first, last) 中每个迭代器的结果。

**可能的实现:**

template<class InputIt, class UnaryFunction>

UnaryFunction for\_each(InputIt first, InputIt last, UnaryFunction f){

for (; first != last; ++first) {

f(\*first);

}

return f; // C++11 起隐式移动}

**例如:**

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> nums{3, 4, 2, 8, 15, 267};

std::for\_each(nums.begin(), nums.end(), [](int &n){ n++; });

这里的function使用了lambda表达式,C++98中不可用,但可以使用boost.lambda

## count/count\_if

template< class InputIt, class T >

typename iterator\_traits<InputIt>::difference\_type

    count( InputIt first, InputIt last, const T &value );

template< class InputIt, class UnaryPredicate >

constexpr typename iterator\_traits<InputIt>::difference\_type

              count\_if( InputIt first, InputIt last, UnaryPredicate p );

* 返回范围 [first, last) 中满足特定判别标准的元素数。

其中模板1根据值来进行判断，模板2使用一元函数进行判断是否满足条件；

例如：

int num\_items1 = std::count(v.begin(), v.end(), target1);

int num\_items3 = std::count\_if(v.begin(), v.end(), [](int i){return i % 3 == 0;});

## mismatch

template< class InputIt1, class InputIt2 >

[std::pair](http://zh.cppreference.com/w/cpp/utility/pair)<InputIt1,InputIt2>  
    mismatch( InputIt1 first1, InputIt1 last1,

              InputIt2 first2 );

* 返回来自二个范围：一个以 [first1, last1) 定义而另一个以 [first2,last2) 定义，的首个不匹配对。若不提供 last2 （重载 (1-4) ），则它指代 first2 + (last1 - first1) 。

**返回值：**

带指向首二个不相等元素的迭代器的 [std::pair](https://zh.cppreference.com/w/cpp/utility/pair" \o "cpp/utility/pair) 。

若比较抵达 last1 时未找到不匹配，则 pair 保有 last1 和来自第二范围的对应迭代器。若第二范围短于第一范围则行为未定义。

## find/find\_if/find\_end/find\_first\_of/adjacend\_find

### find

template< class InputIt, class T >  
InputIt find( InputIt first, InputIt last, const T& value );

* 指向首个满足条件的迭代器，或若找不到这种元素则为 last 。

例如：

vector<string> vecString{“test”, “test1”, “test2”};

vector<string>::iterator itFind = find(vecString.begin(), vecString.end(), “test1”);

if (itFind != vecString.end()) {

return;

}

### find\_if

template< class InputIt, class UnaryPredicate >

InputIt find\_if( InputIt first, InputIt last, UnaryPredicate p );

* 搜索谓词 p 对其返回 true 的元素。

经常可以和boost::bind一起使用

例如：

struct StruTest {

int iValue;

string strValue;

};

StruTest stTest;

vector<StruTest > vecTest;

vector<string>::iterator itFind = find\_if(vecTest.begin(), vecTest.end(), boot::bind(&StruTest::strvalue, \_1) == “aaaa”);

### find\_end

template< class ForwardIt1, class ForwardIt2 >

ForwardIt1 find\_end( ForwardIt1 first, ForwardIt1 last,

                     ForwardIt2 s\_first, ForwardIt2 s\_last );

template< class ForwardIt1, class ForwardIt2, class BinaryPredicate >

ForwardIt1 find\_end( ForwardIt1 first, ForwardIt1 last,

                     ForwardIt2 s\_first, ForwardIt2 s\_last, BinaryPredicate p );

* 在范围 [first, last) 中搜索序列 [s\_first, s\_last) 的最后一次出现。

例：

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> v{1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4};

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int>::iterator result;

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> t1{1, 2, 3};

result = std::find\_end(v.begin(), v.end(), t1.begin(), t1.end());

if (result == v.end()) {

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << "subsequence not found\n";

} else {

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << "last subsequence is at: "

<< [std::distance](http://zh.cppreference.com/w/cpp/iterator/distance)(v.begin(), result) << "\n";

}

### find\_first\_of

template< class ForwardIt1, class ForwardIt2 >

ForwardIt1 find\_first\_of( ForwardIt1 first, ForwardIt1 last,

                          ForwardIt2 s\_first, ForwardIt2 s\_last );

template< class ForwardIt1, class ForwardIt2, class BinaryPredicate >

ForwardIt1 find\_first\_of( ForwardIt1 first, ForwardIt1 last,

                          ForwardIt2 s\_first, ForwardIt2 s\_last, BinaryPredicate p );

* 在范围 [first, last) 中搜索范围 [s\_first, s\_last) 中的任何元素。

例：

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> v{0, 2, 3, 25, 5};

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> t{3, 19, 10, 2};

auto result = std::find\_first\_of(v.begin(), v.end(), t.begin(), t.end());

if (result == v.end()) {

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << "no elements of v were equal to 3, 19, 10 or 2\n";

} else {

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << "found a match at "

<< [std::distance](http://zh.cppreference.com/w/cpp/iterator/distance)(v.begin(), result) << "\n";

}

### adjacent\_find

template< class ForwardIt >  
ForwardIt adjacent\_find( ForwardIt first, ForwardIt last );

* 在范围 [first, last) 中搜索二个相继的等同元素。

例：

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> v1{0, 1, 2, 3, 40, 40, 41, 41, 5};

auto i1 = std::adjacent\_find(v1.begin(), v1.end());

if (i1 == v1.end()) {

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << "no matching adjacent elements\n";

} else {

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << "the first adjacent pair of equal elements at: "

<< [std::distance](http://zh.cppreference.com/w/cpp/iterator/distance)(v1.begin(), i1) << '\n';

}

## search/search\_n

### search

* 搜索范围 [first, last - (s\_last - s\_first)) 中元素子序列 [s\_first, s\_last) 的首次出现

template< class ForwardIt1, class ForwardIt2 >

ForwardIt1 search( ForwardIt1 first, ForwardIt1 last,

                   ForwardIt2 s\_first, ForwardIt2 s\_last );

template< class ForwardIt1, class ForwardIt2, class BinaryPredicate >

ForwardIt1 search( ForwardIt1 first, ForwardIt1 last,

                   ForwardIt2 s\_first, ForwardIt2 s\_last, BinaryPredicate p );

### search\_n

template< class ForwardIt, class Size, class T >  
ForwardIt search\_n( ForwardIt first, ForwardIt last, Size count, const T& value );

* 在范围 [first, last) 中搜索 count 个等同元素的序列，每个都等于给定的值 value 。

**返回值:**

指向范围 [first, last) 中找到的序列起始的迭代器。若找不到这种序列，则返回 last 。

若 count 为零或负，则返回 first 。

# 修改序列的操作

## copy/copy\_backward

### copy

template< class InputIt, class OutputIt >  
OutputIt copy( InputIt first, InputIt last, OutputIt d\_first );

* 复制 [first, last) 所定义的范围中的元素到始于 d\_first 的另一范围。

**返回值：**

指向目标范围中最后复制元素的下个元素的输出迭代器。

例：

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> to\_vector;

std::copy(from\_vector.begin(), from\_vector.end(),

[std::](http://zh.cppreference.com/w/cpp/iterator/back_inserter)**[back\_inserter](http://zh.cppreference.com/w/cpp/iterator/back_inserter)**(to\_vector));

### copy\_backward

template< class BidirIt1, class BidirIt2 >  
BidirIt2 copy\_backward( BidirIt1 first, BidirIt1 last, BidirIt2 d\_last );

* 复制来自 [first, last) 所定义范围的元素，到终于 d\_last 的范围。以逆序复制元素（首先复制末元素），但保持其相对顺序。
* 若 d\_last 在 (first, last] 中则行为未定义。该情况下必须用 [std::copy](https://zh.cppreference.com/w/cpp/algorithm/copy" \o "cpp/algorithm/copy) 取代 std::copy\_backward 。

### **返回值:**

指向最后复制元素的迭代器。

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> from\_vector;

for (int i = 0; i < 10; i++) {

from\_vector.push\_back(i);

}

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> to\_vector(15);

std::copy\_backward(from\_vector.begin(), from\_vector.end(), to\_vector.end());

to\_vector contains: 0 0 0 0 0 0 1 2 3 4 5 6 7 8 9

## fill/fill\_n

### fill

template< class ForwardIt, class T >  
void fill( ForwardIt first, ForwardIt last, const T& value );

* 赋值给定的 value 给 [first, last) 中的元素。

### **返回值**

（无）

例如：

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> v{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

  std::fill(v.begin(), v.end(), -1);

### fill\_n

template< class OutputIt, class Size, class T >  
void fill\_n( OutputIt first, Size count, const T& value );

* 若 count > 0 ，则赋值给定的 value 给始于 的范围的首 count 个元素。否则不做任何事。

### 返回值：

无

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> v1{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

 std::fill\_n(v1.begin(), 5, -1);

## transform

模板<一>

template< class InputIt, class OutputIt, class UnaryOperation >

OutputIt transform( InputIt first1, InputIt last1, OutputIt d\_first,

                    UnaryOperation unary\_op );

应用一元函数 unary\_op 到 [first1, last1) 所定义的范围。

模板<二>

template< class InputIt1, class InputIt2, class OutputIt, class BinaryOperation >

OutputIt transform( InputIt1 first1, InputIt1 last1, InputIt2 first2,

                    OutputIt d\_first, BinaryOperation binary\_op );

* 应用二元函数 binary\_op 到来自二个范围的元素对：一个以 [first1, last1) 定义，而另一个始于 first2

std::transform 应用给定的函数到范围并存储结果于始于 d\_first 的另一范围。

**参数:**

first1, last1: 要变换的第一个元素范围

first2: 要变换的第二个元素的起始

d\_first: 目标范围的起始,可以等于first1或first2

unary\_op: 将要应用的一元函数的签名

binary\_op: 被使用的二元函数对象

**返回值:**

指向最后一个变换的元素的输出迭代器。

可能的实现:

template<class InputIt, class OutputIt, class UnaryOperation>

OutputIt transform(InputIt first1, InputIt last1, OutputIt d\_first,

UnaryOperation unary\_op){

while (first1 != last1) {

\*d\_first++ = unary\_op(\*first1++);

}

template<class InputIt1, class InputIt2,

class OutputIt, class BinaryOperation>

OutputIt transform(InputIt1 first1, InputIt1 last1, InputIt2 first2,

OutputIt d\_first, BinaryOperation binary\_op){

while (first1 != last1) {

\*d\_first++ = binary\_op(\*first1++, \*first2++);

}

return d\_first;}

**注意:**

std::transform 不保证按顺序应用 unary\_op 或 binary\_op 。为按顺序应用函数到数列，或应用修改序列元素的函数，应使用 [std::for\_each](https://zh.cppreference.com/w/cpp/algorithm/for_each" \o "cpp/algorithm/for each) 。

## generate/generate\_n

### generate

template< class ForwardIt, class Generator >  
void generate( ForwardIt first, ForwardIt last, Generator g );

* 以给定函数对象 g 所生成的值赋值范围 [first, last) 中的每个元素。

返回值：无

int f(){

static int i = 1;

return i++;}

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> v(5);

std::generate(v.begin(), v.end(), f);

### generate\_n

template< class OutputIt, class Size, class Generator >  
void generate\_n( OutputIt first, Size count, Generator g );

* 若 count>0 ，则赋值给定函数对象 g 所生成的值给始于 first 的范围的首 count 个元素。否则不做任何事。

## remove/remove\_if/remove\_copy/remove\_copy\_if

### remove/remove\_if

template< class ForwardIt, class T >  
ForwardIt remove( ForwardIt first, ForwardIt last, const T& value );

* 从范围 [first, last) 移除所有满足特定判别标准的元素，并返回范围新结尾的尾后迭代器。
* 返回值：
* 新值范围的尾后迭代器（若它不是 end ，则它指向未指定值，而此迭代器与 end 之间的迭代器所指向的任何值亦然）。

[std::string](http://zh.cppreference.com/w/cpp/string/basic_string) str2 = "Text\n with\tsome \t whitespaces\n\n";

str2.erase(std::remove\_if(str2.begin(),

str2.end(),

[](unsigned char x){return std::isspace(x);}),

### remove\_copy/remove\_copy\_if

template< class InputIt, class OutputIt, class T >

OutputIt remove\_copy( InputIt first, InputIt last, OutputIt d\_first,

                      const T& value );

* 复制来自范围 [first, last) 的元素到始于 d\_first 的另一范围，省略满足特定判别标准的元素。源与目标范围不能重叠。

[std::string](http://zh.cppreference.com/w/cpp/string/basic_string) str = "Text with some spaces";

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << "before: " << str << "\n";

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << "after: ";

std::remove\_copy(str.begin(), str.end(),

[std::ostream\_iterator](http://zh.cppreference.com/w/cpp/iterator/ostream_iterator)<char>([std::cout](http://zh.cppreference.com/w/cpp/io/cout)), ' ');

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << '\n';

## replace/replace\_if/replace\_copy/replace\_copy\_if

### replace/replace\_if

template< class ForwardIt, class T >

void replace( ForwardIt first, ForwardIt last,

              const T& old\_value, const T& new\_value );

* 以 new\_value 替换范围 [first, last) 中所有满足特定判别标准的元素。
* 下列代码首先在 int 的 vector 中以 88 替换 8 的所有出现，然后以 55 替换所有小于 5 的值。

[std::array](http://zh.cppreference.com/w/cpp/container/array)<int, 10> s{5, 7, 4, 2, 8, 6, 1, 9, 0, 3};

std::replace(s.begin(), s.end(), 8, 88);

for (int a : s) {

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << a << " ";

}

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << '\n';

std::replace\_if(s.begin(), s.end(),

[std::bind](http://zh.cppreference.com/w/cpp/utility/functional/bind)([std::less](http://zh.cppreference.com/w/cpp/utility/functional/less)<int>(), std::placeholders::\_1, 5), 55);

for (int a : s) {

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << a << " ";

}

### replace\_copy/replace\_copy\_if

template< class InputIt, class OutputIt, class T >

OutputIt replace\_copy( InputIt first, InputIt last, OutputIt d\_first,

                       const T& old\_value, const T& new\_value );

* 复制来自范围 [first, last) 的所有元素到始于 d\_first 的范围，并以 new\_value 替换所有满足特定判别标准的元素。源与目标范围不能重叠。

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> v{5, 7, 4, 2, 8, 6, 1, 9, 0, 3};

std::replace\_copy\_if(v.begin(), v.end(),

[std::ostream\_iterator](http://zh.cppreference.com/w/cpp/iterator/ostream_iterator)<int>([std::cout](http://zh.cppreference.com/w/cpp/io/cout), " "),

[](int n){return n > 5;}, 99);

## swap/swap\_ranges/iter\_swap

### swap

template< class T >  
void swap( T& a, T& b );

* 交换给定值

int a = 5, b = 3;

// 前

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << a << ' ' << b << '\n';

std::swap(a,b);

### swap\_ranges

template< class ForwardIt1, class ForwardIt2 >

ForwardIt2 swap\_ranges( ForwardIt1 first1, ForwardIt1 last1,

                        ForwardIt2 first2 );

* 在范围 [first1, last1) 和始于 first2 的另一范围间交换元素。
* 返回值：
* 指向始于 first2 的范围中被交换的最末元素后一元素的迭代器。

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> v = {1, 2, 3, 4, 5};

[std::list](http://zh.cppreference.com/w/cpp/container/list)<int> l = {-1, -2, -3, -4, -5};

std::swap\_ranges(v.begin(), v.begin()+3, l.begin());

### iterator\_swap

template< class ForwardIt1, class ForwardIt2 >  
void iter\_swap( ForwardIt1 a, ForwardIt2 b );

* 交换给定的迭代器所指向的元素的值。

例如：

template<class ForwardIt>void selection\_sort(ForwardIt begin, ForwardIt end){

for (ForwardIt i = begin; i != end; ++i)

std::iter\_swap(i, [std::min\_element](http://zh.cppreference.com/w/cpp/algorithm/min_element)(i, end));}

## reverse/reverse\_copy

### reverse

template< class BidirIt >  
void reverse( BidirIt first, BidirIt last );

* 反转 [first, last) 范围中的元素顺序
* 表现如同应用 [std::iter\_swap](https://zh.cppreference.com/w/cpp/algorithm/iter_swap" \o "cpp/algorithm/iter swap) 到对于非负 i < (last-first)/2 的每对迭代器 first+i, (last-i) - 1

例如：

int a[] = {4, 5, 6, 7};

std::reverse([std::begin](http://zh.cppreference.com/w/cpp/iterator/begin)(a), [std::end](http://zh.cppreference.com/w/cpp/iterator/end)(a));

### reverse\_copy

template< class BidirIt, class OutputIt >  
OutputIt reverse\_copy( BidirIt first, BidirIt last, OutputIt d\_first );

* 复制来自范围 [first, last) 的元素到始于 d\_first 的新范围，使得新范围中元素以逆序排列。

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> v({1,2,3});

for (const auto& value : v) {

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << value << " ";

}

[std::cout](http://zh.cppreference.com/w/cpp/io/cout) << '\n';

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> destination(3);

std::reverse\_copy([std::begin](http://zh.cppreference.com/w/cpp/iterator/begin)(v), [std::end](http://zh.cppreference.com/w/cpp/iterator/end)(v), [std::begin](http://zh.cppreference.com/w/cpp/iterator/begin)(destination));

## rotate/rotate\_copy

### rotate

template< class ForwardIt >  
void      rotate( ForwardIt first, ForwardIt n\_first, ForwardIt last );

* 进行元素范围上的左旋转。
* 具体而言， std::rotate 交换范围 [first, last) 中的元素，方式满足元素 n\_first 成为新范围的首个元素，而 n\_first - 1 成为最后元素。

 此函数的前提条件是 [first, n\_first) 和 [n\_first, last) 为合法范围。

### rotate\_copy

template< class ForwardIt, class OutputIt >

OutputIt rotate\_copy( ForwardIt first, ForwardIt n\_first,

                      ForwardIt last, OutputIt d\_first );

* 从范围 [first, last) 复制元素到始于 d\_first 的另一范围，使得 n\_first 成为新范围的首元素，而 n\_first - 1 成为末元素。

## random\_shuffle

## unique

template< class ForwardIt >  
ForwardIt unique( ForwardIt first, ForwardIt last );

* 从来自范围 [first, last) 的相继等价元素组消除首元素外的元素，并返回范围的新逻辑结尾的尾后迭代器。
* 通过用覆写要被擦除的元素的方式迁移范围中的元素进行移除。保持剩余元素的相对顺序，且不更改容器的物理大小。指向范围的新逻辑结尾和物理结尾之间元素的迭代器仍然可解引用，但元素自身拥有未指定值。调用 unique 典型地后随调用容器的 erase 方法，它擦除未指定值并减小容器的物理大小，以匹配其新的逻辑大小。

// 移除重复元素

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> v{1,2,3,1,2,3,3,4,5,4,5,6,7};

[std::sort](http://zh.cppreference.com/w/cpp/algorithm/sort)(v.begin(), v.end()); // 1 1 2 2 3 3 3 4 4 5 5 6 7

auto last = std::unique(v.begin(), v.end());

// v 现在保有 {1 2 3 4 5 6 7 x x x x x x} ，其中 x 不确定

v.erase(last, v.end());

## unique\_copy

template< class InputIt, class OutputIt >

OutputIt unique\_copy( InputIt first, InputIt last,

                      OutputIt d\_first );

* 从范围 [first, last) 复制元素到始于 d\_first 的另一范围，使得无连续的相等元素。只复制每组等价元素的首元素。

# 划分操作

## partition

template< class BidirIt, class UnaryPredicate >  
BidirIt partition( BidirIt first, BidirIt last, UnaryPredicate p );

* 重排序范围 [first, last) 中的元素，使得谓词 p 对其返回 true 的元素前于谓词 p 对其返回 false 的元素。不保持相对顺序。

## stable\_partition

template< class BidirIt, class UnaryPredicate >  
BidirIt stable\_partition( BidirIt first, BidirIt last, UnaryPredicate p );

* 重排序范围 [first, last) 中的元素，使得所有谓词 p 对其返回 true 的元素先于谓词 p 对其返回 false 的元素。保持元素的相对顺序。

# 排序操作

## sort

* 在很多应用中，排序都是至关重要的，而且很多 [STL](http://c.biancheng.net/stl/" \t "http://c.biancheng.net/view/_blank) 算法也只适用于有序对象序列。定义在 algorithm 头文件中的函数模板 sort<Iter>() 默认会将元素段排成升序，这也就意味着排序的对象的类型需要支持 < 运算符。
* 函数模板 sort<Iter>() 的类型参数 Iter 是元素段元素对应的迭代器类型，而且它们必须支持随机访问迭代器。这表明 sort() 算法只能对提供随机访问迭代器的容器中的元素进行排序，也说明 sort() 只能接受 array、vector、deque 或标准数组中的元素。

升序：std::sort(++std::begin(numbers),--std::end(numbers));

降序：std::sort(std::begin(numbers), std::end(numbers), std::greater<>());

## partial\_sort

template< class RandomIt >  
void partial\_sort( RandomIt first, RandomIt middle, RandomIt last );

template< class RandomIt, class Compare >

void partial\_sort( RandomIt first, RandomIt middle, RandomIt last,

                   Compare comp );

* 重排元素，使得范围 [first, middle) 含有范围 [first, last) 中已排序的 middle - first 个最小元素。
* 不保证保持相等的元素顺序。范围 [middle, last) 中剩余的元素顺序未指定。

## partial\_sort\_copy

## stable\_sort

template< class RandomIt >  
void stable\_sort( RandomIt first, RandomIt last );

* 以升序排序范围 [first, last) 中的元素。保证保持等价元素的顺序。

## nth\_element

# 二分搜索操作（在已排序基础上）

## lower\_bound

template< class ForwardIt, class T >  
ForwardIt lower\_bound( ForwardIt first, ForwardIt last, const T& value );

template< class ForwardIt, class T, class Compare >  
ForwardIt lower\_bound( ForwardIt first, ForwardIt last, const T& value, Compare comp );

* 返回指向范围 [first, last) 中首个不小于（即大于或等于） value 的元素的迭代器，或若找不到这种元素则返回 last 。
* 范围 [first, last) 必须已相对于表达式 element < value 或 comp(element, value) 划分，即所有令该表达式为 true 的元素必须前趋所有令该表达式为 false 的元素。完全排序的范围满足此判别标准。
* 第一版本用 operator< 比较元素，第二版本用给定的比较函数 comp 。

## upper\_bound

template< class ForwardIt, class T >  
ForwardIt upper\_bound( ForwardIt first, ForwardIt last, const T& value );

template< class ForwardIt, class T, class Compare >  
ForwardIt upper\_bound( ForwardIt first, ForwardIt last, const T& value, Compare comp );

* 返回指向范围 [first, last) 中首个大于 value 的元素的迭代器，或若找不到这种元素则返回 last 。
* 范围 [first, last) 必须已相对于表达式 !(value < element) 或 !comp(value, element) 划分，即所有令此表达式为 true 的元素必须前趋所有令此表达式为 false 的元素。完全排序的范围满足此判别标准。
* 第一版本用 operator< 比较元素，第二版本用给定的比较函数 comp 。

## binary\_search

template< class ForwardIt, class T >  
bool binary\_search( ForwardIt first, ForwardIt last, const T& value );

template< class ForwardIt, class T, class Compare >  
bool binary\_search( ForwardIt first, ForwardIt last, const T& value, Compare comp );

* 检查等价于 value 的元素是否出现于范围 [first, last) 中。
* 对于要成功的 std::binary\_search ，范围 [first, last) 必须至少相对于 value 部分有序，即它必须满足下列所有要求：
* 已相对 element < value 或 comp(element, value) 划分
* 已相对 !(value < element) 或 !comp(value, element) 划分（即所有令此表达式为 true 的元素必须前趋所有令此表达式为 false 的元素）
* 对于所有元素，若 element < value 或 comp(element, value) 为 true ，则 !(value < element) 或 !comp(value, element) 亦为 true
* 完全排序的范围满足这些判别标准。
* 第一版本用 operator< 比较元素，第二版本用给定的比较函数 comp 。

## equal\_range

template< class ForwardIt, class T >

[std::pair](http://zh.cppreference.com/w/cpp/utility/pair)<ForwardIt,ForwardIt>   
    equal\_range( ForwardIt first, ForwardIt last,

                 const T& value );

template< class ForwardIt, class T, class Compare >

[std::pair](http://zh.cppreference.com/w/cpp/utility/pair)<ForwardIt,ForwardIt>   
    equal\_range( ForwardIt first, ForwardIt last,

                 const T& value, Compare comp );

* 返回范围 [first, last) 中含有所有等价于 value 的元素的范围。
* 范围 [first, last) 必须至少相对于 value 有序，即它必须满足下列所有要求：
* 已相对 element < value 或 comp(element, value) 划分（即所有令该表达式为 true 的元素必须前趋所有令此表达式为 false 的元素）
* 已相对 !(value < element) 或 !comp(value, element) 划分
* 对于所有元素，若 element < value 或 comp(element, value) 为 true ，则 !(value < element) 或 !comp(value, element) 亦为 true
* 完全排序的范围满足这些判别标准。
* 以二个迭代器定义返回的范围，一个指向首个不小于 value 的元素，而另一个指向首个大于 value 的元素。可替而用 [std::lower\_bound()](https://zh.cppreference.com/w/cpp/algorithm/lower_bound" \o "cpp/algorithm/lower bound) 获得第一迭代器，用 [std::upper\_bound()](https://zh.cppreference.com/w/cpp/algorithm/upper_bound" \o "cpp/algorithm/upper bound) 获得第二迭代器。
* 第一版本用 operator< 比较元素，第二版本用给定的比较函数 comp 。

# 集合操作（在已排序基础上）

## merge

template< class InputIt1, class InputIt2, class OutputIt >

OutputIt merge( InputIt1 first1, InputIt1 last1,  
                InputIt2 first2, InputIt2 last2,

                OutputIt d\_first );

template< class InputIt1, class InputIt2, class OutputIt, class Compare>

OutputIt merge( InputIt1 first1, InputIt1 last1,  
                InputIt2 first2, InputIt2 last2,

                OutputIt d\_first, Compare comp );

* 归并二个已排序范围 [first1, last1) 和 [first2, last2) 到始于 d\_first 的一个已排序范围中。
* 1) 用 operator< 比较元素。
* 3) 用给定的二元比较函数 comp 比较元素。
* 2,4) 同 (1,3) ，但按照 policy 执行。这些重载仅若 [std::is\_execution\_policy\_v](http://zh.cppreference.com/w/cpp/algorithm/is_execution_policy)<[std::decay\_t](http://zh.cppreference.com/w/cpp/types/decay)<ExecutionPolicy>> 为 true 才参与重载决议。
* 对于元范围中的等价元素，来自第一范围的元素（保持其原顺序）先于来自第二范围的元素（保持其原顺序）。
* 若目标范围与输入范围之一重叠，则行为未定义（输入范围可相互重叠）。

## inplace\_merge

template< class BidirIt >  
void inplace\_merge( BidirIt first, BidirIt middle, BidirIt last );

* 归并二个相继的已排序范围 [first, middle) 及 [middle, last) 为一个已排序范围 [first, last) 。对于原先二个范围中的等价元素。来自第一范围的元素（保持其原顺序）先于来自第二范围的元素（保持其原顺序）。
* 1) 用 operator< 比较元素，且范围必须对于同关系已排序。
* 3) 用给定的二元比较函数 comp 比较元素，且范围必须对于同关系已排序。
* 2,4) 同 (1,3) ，但按照 policy 执行。此重载仅若 [std::is\_execution\_policy\_v](http://zh.cppreference.com/w/cpp/algorithm/is_execution_policy)<[std::decay\_t](http://zh.cppreference.com/w/cpp/types/decay)<ExecutionPolicy>>为 true 才参与重载决议。

## includes

template< class InputIt1, class InputIt2 >

bool includes( InputIt1 first1, InputIt1 last1,

               InputIt2 first2, InputIt2 last2 );

template< class InputIt1, class InputIt2, class Compare >

bool includes( InputIt1 first1, InputIt1 last1,

               InputIt2 first2, InputIt2 last2, Compare comp );

* 若已排序范围 [first2, last2) 是已排序范围 [first1, last1) 的[子序列](https://en.wikipedia.org/wiki/Subsequence" \o "enwiki:Subsequence)则返回 true 。（子序列不必相接。）
* 1) 两个范围都必须以 operator< 排序。
* 3) 两个范围都必须以给定的比较函数 comp 排序。

## set\_difference

template< class InputIt1, class InputIt2, class OutputIt >

OutputIt set\_difference( InputIt1 first1, InputIt1 last1,  
                         InputIt2 first2, InputIt2 last2,

                         OutputIt d\_first );

template< class InputIt1, class InputIt2,

          class OutputIt, class Compare >  
OutputIt set\_difference( InputIt1 first1, InputIt1 last1,  
                         InputIt2 first2, InputIt2 last2,

                         OutputIt d\_first, Compare comp );

* 复制来自已排序范围 [first1, last1) 并且不在已排序范围 [first2, last2) 中找到的元素到始于 d\_first 的范围。
* 结果范围亦已排序。单独对待等价元素，即若某元素在 [first1, last1) 中找到 n 次而在 [first2, last2) 中找到 m 次，则它将被准确复制 [std::max](http://zh.cppreference.com/w/cpp/algorithm/max)(m-n, 0) 次到 d\_first 。结果范围不能与任一输入范围重叠。
* 1) 用 operator< 比较元素，而范围必须对于相同者排序。
* 3) 用给定的二元比较函数 comp 比较元素，而范围必须对于相同者排序。

## set\_intersection

template< class InputIt1, class InputIt2, class OutputIt >

OutputIt set\_intersection( InputIt1 first1, InputIt1 last1,  
                           InputIt2 first2, InputIt2 last2,

                           OutputIt d\_first );

template< class InputIt1, class InputIt2,

          class OutputIt, class Compare >  
OutputIt set\_intersection( InputIt1 first1, InputIt1 last1,  
                           InputIt2 first2, InputIt2 last2,

                           OutputIt d\_first, Compare comp );

* 构造始于 d\_first ，由在两个已排序范围 [first1, last1) 与 [first2, last2) 中都找到的元素构成的已排序范围。若某元素在 [first1, last1) 中找到 m 而在 [first2, last2) 中找到 n 次，则从第一范围复制首 [std::min](http://zh.cppreference.com/w/cpp/algorithm/min)(m, n) 个元素到目标范围。保持等价元素的顺序。结果范围不能与任一输入范围重叠。
* @1用 operator< 比较元素，而范围必须相对同一标准已排序。
* 3) 用给定的二元比较函数 comp 比较元素，而范围必须相对同一标准已排序。

## set\_symmetric\_difference

template< class InputIt1, class InputIt2, class OutputIt >

OutputIt set\_symmetric\_difference( InputIt1 first1, InputIt1 last1,  
                                   InputIt2 first2, InputIt2 last2,

                                   OutputIt d\_first );

template< class InputIt1, class InputIt2,

          class OutputIt, class Compare >  
OutputIt set\_symmetric\_difference( InputIt1 first1, InputIt1 last1,  
                                   InputIt2 first2, InputIt2 last2,

                                   OutputIt d\_first, Compare comp );

* 计算二个已排序范围的对称差：复制在任一范围中，但不在两个范围中找到的元素，到始于 d\_first 的范围。结果范围亦为已排序。
* 若某元素在 [first1, last1) 中找到 m 次而在 [first2, last2) 中找到 n 次，则将准确复制它 std::abs(m-n) 次到 d\_first 。若 m>n ，则从[first1,last1) 复制末 m-n 个元素，否则从 [first2,last2) 复制末 n-m 个元素。结果范围不能与任一输入范围重叠。
* 1) 用 operator< 比较元素，而范围必须相对于相同标准已排序。
* 3) 用给定的二元比较函数 comp 比较元素，而范围必须相对相同标准已排序。

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> v1{1,2,3,4,5,6,7,8 };

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> v2{ 5, 7, 9,10};

[std::sort](http://zh.cppreference.com/w/cpp/algorithm/sort)(v1.begin(), v1.end());

[std::sort](http://zh.cppreference.com/w/cpp/algorithm/sort)(v2.begin(), v2.end());

[std::vector](http://zh.cppreference.com/w/cpp/container/vector)<int> v\_symDifference;

std::set\_symmetric\_difference(

v1.begin(), v1.end(),

v2.begin(), v2.end(),

[std::back\_inserter](http://zh.cppreference.com/w/cpp/iterator/back_inserter)(v\_symDifference));

输出：

1 2 3 4 6 8 9 10

## set\_union

template< class InputIt1, class InputIt2, class OutputIt >

OutputIt set\_union( InputIt1 first1, InputIt1 last1,  
                    InputIt2 first2, InputIt2 last2,

                    OutputIt d\_first );

template< class InputIt1, class InputIt2,

          class OutputIt, class Compare >  
OutputIt set\_union( InputIt1 first1, InputIt1 last1,  
                    InputIt2 first2, InputIt2 last2,

                    OutputIt d\_first, Compare comp );

* 构造始于 d\_first ，由存在于一或两个已排序范围 [first1, last1) 和 [first2, last2) 中的所有元素构成的已排序范围。
* 若某元素在 [first1, last1) 中找到 m 次并在 [first2, last2) 中找到 n 次，则将从 [first1, last1) 保持顺序复制全部 m个元素到 d\_first ，并从 [first2, last2) 保持顺序复制 [std::max](http://zh.cppreference.com/w/cpp/algorithm/max)(n-m, 0) 个元素到 d\_first 。
* 结果范围不能与任一输入范围重叠。
* 1) 用 operator< 比较元素，而范围必须相对于相同标准已排序。
* 3) 用给定的二元比较函数 comp 比较元素，而范围必须相对相同标准已排序。

# 堆操作

is\_heap

make\_heap

push\_heap

pop\_heap

sort\_heap

# 最小、最大操作

## max

template< class T >   
const T& max( const T& a, const T& b );

template< class T, class Compare >  
const T& max( const T& a, const T& b, Compare comp );

* 返回给定值中的较大者。
* 1-2) 返回 a 与 b 的较大者。

## max\_element

template< class ForwardIt >   
ForwardIt max\_element(ForwardIt first, ForwardIt last );

template< class ForwardIt, class Compare >  
ForwardIt max\_element(ForwardIt first, ForwardIt last, Compare comp );

* 寻找范围 [first, last) 中的最大元素。
* 1) 用 operator< 比较元素。
* 3) 用给定的二元比较函数 comp 比较元素。
* 返回值：
* 指向范围 [first, last) 中最大元素的迭代器。若范围中有多个元素等价于最大元素，则返回指向首个这种元素的迭代器。若范围为空则返回 last 。

## min

## min\_element

# 比较操作

## equal

template< class InputIt1, class InputIt2 >

bool equal( InputIt1 first1, InputIt1 last1,

            InputIt2 first2 );

template< class InputIt1, class InputIt2, class BinaryPredicate >

bool equal( InputIt1 first1, InputIt1 last1,

            InputIt2 first2, BinaryPredicate p );

* 1,3) 如果范围 [first1, last1) 和范围 [first2, first2 + (last1 - first1)) 相等，返回 true ，否则返回 false
* 5,7) 如果范围 [first1, last1) 和范围 [first2, last2) 相等，返回 true ，否则返回 false
* 2,4,6,8) 同 (1,3,5,7) ，但按照 policy 执行。这些重载仅若 [std::is\_execution\_policy\_v](http://zh.cppreference.com/w/cpp/algorithm/is_execution_policy)<[std::decay\_t](http://zh.cppreference.com/w/cpp/types/decay)<ExecutionPolicy>> 为 true 才参与重载决议。
* 两个范围相等的条件是：对于范围 [first1, last1) 内的每个迭代器 i ， \*i 等于 \*(first2 + (i - first1)) 。重载形式 (1,2,5,6) 用 operator== 判定两个元素是否相等，而重载形式 (3,4,7,8) 用给定的谓词函数。
* 返回值
* 5-8) 如果范围 [first1, last1) 的长度不等于范围 [first2, last2) 的长度则返回 false 。
* 如果两个范围内的元素都相等，则返回 true 。
* 否则返回 false 。
* 注意：
* std::equal 不可应用到由 [std::unordered\_set](https://zh.cppreference.com/w/cpp/container/unordered_set" \o "cpp/container/unordered set) 、 [std::unordered\_multiset](https://zh.cppreference.com/w/cpp/container/unordered_multiset" \o "cpp/container/unordered multiset) 、 [std::unordered\_map](https://zh.cppreference.com/w/cpp/container/unordered_map" \o "cpp/container/unordered map) 或 [std::unordered\_multimap](https://zh.cppreference.com/w/cpp/container/unordered_multimap" \o "cpp/container/unordered multimap) 的迭代器构成的范围，因为即使此类容器存储相同的元素，在容器内元素存储的顺序也可能不同。
* 比较整个容器是否相等时，针对该容器的 operator== 重载通常是更好的选择。

## lexicographical\_compare

template< class InputIt1, class InputIt2 >

bool lexicographical\_compare( InputIt1 first1, InputIt1 last1,

                              InputIt2 first2, InputIt2 last2 );

template< class InputIt1, class InputIt2, class Compare >

bool lexicographical\_compare( InputIt1 first1, InputIt1 last1,  
                              InputIt2 first2, InputIt2 last2,

                              Compare comp );

* 检查第一个范围 [first1, last1) 是否按字典序小于第二个范围 [first2, last2) 。

# 排列操作

## next\_permutation

template< class BidirIt >  
bool next\_permutation( BidirIt first, BidirIt last );

template< class BidirIt, class Compare >  
bool next\_permutation( BidirIt first, BidirIt last, Compare comp );

* 变换范围 [first, last) 为来自所有按相对于 operator< 或 comp 的字典序的下个排列。若这种排列存在则返回 true ，否则变换范围为首个排列（如同用 std::sort(first, last) ）并返回 false 。
* 返回值：
* 若新排列按字典序大于旧者则为 true 。若抵达最后重排并重置范围为首个排列则为 false 。

## prev\_permutation

template< class BidirIt >  
bool prev\_permutation( BidirIt first, BidirIt last);

template< class BidirIt, class Compare >  
bool prev\_permutation( BidirIt first, BidirIt last, Compare comp);

* 变换范围 [first, last) 为来自于相对于 operator< 或 comp 的字典序的所有排列集合的上个排列。若这种排列存在则返回 true ，否则变换范围为末排列（如同用 std::sort(first, last); std::reverse(first, last); ）并返回 false 。
* 返回值
* 若新排列按字典序前趋旧排列则为 true 。若抵达首个排列并重置范围为最末排列则为 false 。

# 数值运算

定义于头文件<numeric>

## accumulate

template< class InputIt, class T >  
T accumulate( InputIt first, InputIt last, T init );

template< class InputIt, class T, class BinaryOperation >

T accumulate( InputIt first, InputIt last, T init,

              BinaryOperation op );

* 计算给定值 init 与给定范围 [first, last) 中元素的和。第一版本用 operator+ ，第二版本用二元函数 op 求和元素，均将[std::move](https://zh.cppreference.com/w/cpp/utility/move" \o "cpp/utility/move) 应用到其左侧运算数 (C++20 起)。

## inner\_product

template< class InputIt1, class InputIt2, class T >

T inner\_product( InputIt1 first1, InputIt1 last1,

                 InputIt2 first2, T init );

template<class InputIt1, class InputIt2, class T,

         class BinaryOperation1, class BinaryOperation2>   
T inner\_product( InputIt1 first1, InputIt1 last1,  
                 InputIt2 first2, T init,  
                 BinaryOperation1 op1,

                 BinaryOperation2 op2 );

* 计算内积（即积之和）或在范围 [first1, last1) 和始于 first2 的范围上进行有序映射/规约操作。

## adjacent\_difference

template< class InputIt, class OutputIt >

OutputIt adjacent\_difference( InputIt first, InputIt last,

                              OutputIt d\_first );

* 计算 [first, last) 范围中每对相邻元素的第二个和第一个的差，并写入它们到始于 d\_first + 1 的范围。写入不修改的 \*first副本到 \*d\_first 。

## partial\_sum

template< class InputIt, class OutputIt >  
OutputIt partial\_sum( InputIt first, InputIt last, OutputIt d\_first );

template< class InputIt, class OutputIt, class BinaryOperation >

OutputIt partial\_sum( InputIt first, InputIt last, OutputIt d\_first,

                      BinaryOperation op );

* 计算范围 [first, last) 的子范围中元素的部分和，并写入到始于 d\_first 的范围。第一版本用 operator+ ，第二版本用给定的二元函数 op 对元素求和，均将 [std::move](https://zh.cppreference.com/w/cpp/utility/move" \o "cpp/utility/move) 应用到其左侧运算数 (C++20 起)。

# 未初始化内存上的操作

uninitialized\_

## C库

定义于头文件<cstdlib>

## qsort

void qsort( void \*ptr, [std::size\_t](http://zh.cppreference.com/w/cpp/types/size_t) count, [std::size\_t](http://zh.cppreference.com/w/cpp/types/size_t) size, /\*compare-pred\*/\* comp );  
void qsort( void \*ptr, [std::size\_t](http://zh.cppreference.com/w/cpp/types/size_t) count, [std::size\_t](http://zh.cppreference.com/w/cpp/types/size_t) size, /\*c-compare-pred\*/\* comp );extern "C++" using /\*compare-pred\*/ = int(const void\*, const void\*); // 仅为说明  
extern "C" using /\*c-compare-pred\*/ = int(const void\*, const void\*); // 仅为说明

* 以升序排序 ptr 所指向的给定数组。数组含 count 个 size 字节大小的元素。用 comp 所指向的函数比较对象。
* 若 comp 指示二个元素等价，则其顺序未指定。
* 注意：
* 无关乎名称， C++ 、 C 及 POSIX 都不要求此函数用[快速排序](https://en.wikipedia.org/wiki/Quicksort" \o "enwiki:Quicksort)实现或作出任何复杂度和稳定性保证。
* 数组元素必须是[平凡类型 (TrivialType)](https://zh.cppreference.com/w/cpp/named_req/TrivialType" \o "cpp/named req/TrivialType)，否则行为未定义。
* C++ 标准库提供的二个重载不同，因为参数 comp 类型有别（[语言链接](https://zh.cppreference.com/w/cpp/language/language_linkage" \o "cpp/language/language linkage)是类型的一部分）。

## bsearch

void\* bsearch( const void\* key, const void\* ptr, [std::size\_t](http://zh.cppreference.com/w/cpp/types/size_t) count,

[std::size\_t](http://zh.cppreference.com/w/cpp/types/size_t) size, /\*compare-pred\*/\* comp );  
void\* bsearch( const void\* key, const void\* ptr, [std::size\_t](http://zh.cppreference.com/w/cpp/types/size_t) count,

[std::size\_t](http://zh.cppreference.com/w/cpp/types/size_t) size, /\*c-compare-pred\*/\* comp );

extern "C++" using /\*compare-pred\*/ = int(const void\*, const void\*); // 仅为说明  
extern "C" using /\*c-compare-pred\*/ = int(const void\*, const void\*); // 仅为说明

* 在 ptr 所指向的数组中寻找等于 key 所指向的元素的元素。数组含有 count 个 size 字节的元素，且必须相对于 key 所指向的对象划分，即所有比较小于它的元素必须先出现于比较等于它的元素，而比较等于它者必须先出现于所有比较大于该关键对象的元素。完全排序的数组满足这些要求。用 comp 所指的函数比较数组元素。
* 若数组未以按照 comp 所用同样原则，相对于 key 升序划分，则行为未定义。
* 若数组含有数个 comp 会指示等于被搜索元素的元素，则函数返回哪个元素是未指定的。
* 返回值
* 指向找到的元素的指针，或若找不到元素则为空指针。
* 注意
* 无关乎名称， C 或 POSIX 标准都不要求此函数用二分查找实现，或进行任何复杂度保证。
* C++ 标准库提供的二个重载不同，因为参数 comp 类型有别（[语言链接](https://zh.cppreference.com/w/cpp/language/language_linkage" \o "cpp/language/language linkage)是类型的一部分）。