Reference   <algorithm>   lower_bound

C++
- Information
- Tutorials
- Reference
- Articles
- Forum

Reference
- *C library:*
- *Containers:*
- *Input/Output:*
- *Multi-threading:*
- *Other:*
  - **<algorithm>**
  - **<bitset>**
  - **<chrono>**
  - **<codecvt>**
  - **<complex>**
  - **<exception>**
  - **<functional>**
  - **<initializer_list>**
  - **<iterator>**
  - **<limits>**
  - **<locale>**
  - **<memory>**
  - **<new>**
  - **<numeric>**
  - **<random>**
  - **<ratio>**
  - **<regex>**
  - **<stdexcept>**
  - **<string>**
  - **<system_error>**
  - **<tuple>**
  - **<typeindex>**
  - **<typeinfo>**
  - **<type_traits>**
  - **<utility>**
  - **<valarray>**

<algorithm>

function template

std::**lower_bound**   <algorithm>

default (1)	`template <class ForwardIterator, class T>`
	`ForwardIterator lower_bound (ForwardIterator first, ForwardIterator last,`
	`                               const T& val);`

custom (2)	`template <class ForwardIterator, class T, class Compare>`
	`ForwardIterator lower_bound (ForwardIterator first, ForwardIterator last,`
	`                               const T& val, Compare comp);`

Return iterator to lower bound

Returns an iterator pointing to the first element in the range [first,last) which does not compare less than *val*.

The elements are compared using `operator<` for the first version, and *comp* for the second. The elements in the range shall already be sorted according to this same criterion (`operator<` or *comp*), or at least partitioned with respect to *val*.

The function optimizes the number of comparisons performed by comparing non-consecutive elements of the sorted range, which is specially efficient for random-access iterators.

Unlike upper_bound, the value pointed by the iterator returned by this function may also be equivalent to *val*, and not only greater.

The behavior of this function template is equivalent to:

```
1  template <class ForwardIterator, class T>
2    ForwardIterator lower_bound (ForwardIterator first, ForwardIterator last, const T& val)
3  {
4    ForwardIterator it;
5    iterator_traits<ForwardIterator>::difference_type count, step;
6    count = distance(first,last);
7    while (count>0)
8    {
9      it = first; step=count/2; advance (it,step);
10     if (*it<val) {                  // or: if (comp(*it,val)), for version (2)
11       first=++it;
12       count-=step+1;
13     }
14     else count=step;
15   }
16   return first;
17 }
```

Parameters

first, last
>   Forward iterators to the initial and final positions of a sorted (or properly partitioned) sequence. The range used is [first,last), which contains all the elements between *first* and *last*, including the element pointed by *first* but not the element pointed by *last*.

val
>   Value of the lower bound to search for in the range.
>   For *(1)*, T shall be a type supporting being compared with elements of the range [first,last) as the right-hand side operand of `operator<`.

comp
>   Binary function that accepts two arguments (the first of the type pointed by `ForwardIterator`, and the second, always *val*), and returns a value convertible to bool. The value returned indicates whether the first argument is considered to go before the second.
>   The function shall not modify any of its arguments.
>   This can either be a function pointer or a function object.

Return value

An iterator to the lower bound of *val* in the range.
If all the element in the range compare less than *val*, the function returns *last*.

Example

```
1  // lower_bound/upper_bound example
2  #include <iostream>     // std::cout
3  #include <algorithm>    // std::lower_bound, std::upper_bound, std::sort
4  #include <vector>       // std::vector
5
6  int main () {
7    int myints[] = {10,20,30,30,20,10,10,20};
8    std::vector<int> v(myints,myints+8);           // 10 20 30 30 20 10 10 20
9
10   std::sort (v.begin(), v.end());                // 10 10 10 20 20 20 30 30
11
12   std::vector<int>::iterator low,up;
13   low=std::lower_bound (v.begin(), v.end(), 20); //          ^
14   up= std::upper_bound (v.begin(), v.end(), 20); //                   ^
15
16   std::cout << "lower_bound at position " << (low- v.begin()) << '\n';
17   std::cout << "upper_bound at position " << (up - v.begin()) << '\n';
18
```

```
19   return 0;
20 }
```

Output:
```
lower_bound at position 3
upper_bound at position 6
```

Complexity

On average, logarithmic in the distance between *first* and *last*: Performs approximately $\log_2(N)+1$ element comparisons (where N is this distance).
On *non-random-access iterators*, the iterator advances produce themselves an additional linear complexity in N on average.

Data races

The objects in the range [`first,last`) are accessed.

Exceptions

Throws if either an element comparison or an operation on an iterator throws.
Note that invalid arguments cause *undefined behavior*.

See also

upper_bound	Return iterator to upper bound (function template)
equal_range	Get subrange of equal elements (function template)
binary_search	Test if value exists in sorted sequence (function template)
min_element	Return smallest element in range (function template)