Search: Go

Reference <algorithm> lower_bound

Not logged in

register log in

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>
adjacent_find
all_of
any_of
binary_search
copy
copy_backward
copy_if
copy_n
count
count_if
equal
equal_range
fill
fill_n
find
find_end
find_first_of
find_if
find_if_not
for_each
generate
generate_n
includes
inplace_merge
is_heap
is_heap_until
is_partitioned
is_permutation
is_sorted
is_sorted_until
iter_swap
lexicographical_compare
lower_bound
make_heap
max
max_element
merge
min
minmax
minmax_element
min_element
mismatch
move
move_backward
next_permutation
none_of
nth_element

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:

```
template <class ForwardIterator, class T>
  ForwardIterator lower_bound (ForwardIterator first, ForwardIterator last, const T& val)
{
  ForwardIterator it;
  iterator_traits<ForwardIterator>::difference_type count, step;
  count = distance(first,last);
  while (count>0)
  {
    it = first; step=count/2; advance (it,step);
    if (*it<val) {                 // or: if (comp(*it,val)), for version (2)
      first=++it;
      count-=step+1;
    }
    else count=step;
  }
  return first;
}
```

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

```
// lower_bound/upper_bound example
#include <iostream>     // std::cout
#include <algorithm>    // std::lower_bound, std::upper_bound, std::sort
#include <vector>       // std::vector

int main () {
  int myints[] = {10,20,30,30,20,10,10,20};
  std::vector<int> v(myints,myints+8);           // 10 20 30 30 20 10 10 20

  std::sort (v.begin(), v.end());                // 10 10 10 20 20 20 30 30

  std::vector<int>::iterator low,up;
  low=std::lower_bound (v.begin(), v.end(), 20); //          ^
  up= std::upper_bound (v.begin(), v.end(), 20); //                   ^

  std::cout << "lower_bound at position " << (low- v.begin()) << '\n';
  std::cout << "upper_bound at position " << (up - v.begin()) << '\n';

  return 0;
}
```

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)