Search: Go

Not logged in

Reference <algorithm> stable_sort 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
partial_sort
partial_sort_copy
partition
partition_copy
partition_point
pop_heap

function template

std::**stable_sort**

`<algorithm>`

```
template <class RandomAccessIterator>
  void stable_sort ( RandomAccessIterator first, RandomAccessIterator last );

template <class RandomAccessIterator, class Compare>
  void stable_sort ( RandomAccessIterator first, RandomAccessIterator last,
                     Compare comp );
```

Sort elements preserving order of equivalents

Sorts the elements in the range `[first,last)` into ascending order, like sort, but stable_sort preserves the relative order of the elements with equivalent values.

The elements are compared using `operator<` for the first version, and *comp* for the second.

Parameters

`first, last`
: Random-access iterators to the initial and final positions of the sequence to be sorted. 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*.
`RandomAccessIterator` shall point to a type for which `swap` is properly defined and which is both *move-constructible* and *move-assignable*.

`comp`
: Binary function that accepts two elements in the range as arguments, and returns a value convertible to `bool`. The value returned indicates whether the element passed as first argument is considered to go before the second in the specific *strict weak ordering* it defines.
The function shall not modify any of its arguments.
This can either be a function pointer or a function object.

Return value

none

Example

```
// stable_sort example
#include <iostream>     // std::cout
#include <algorithm>    // std::stable_sort
#include <vector>       // std::vector

bool compare_as_ints (double i,double j)
{
  return (int(i)<int(j));
}

int main () {
  double mydoubles[] = {3.14, 1.41, 2.72, 4.67, 1.73, 1.32, 1.62, 2.58};

  std::vector<double> myvector;

  myvector.assign(mydoubles,mydoubles+8);

  std::cout << "using default comparison:";
  std::stable_sort (myvector.begin(), myvector.end());
  for (std::vector<double>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  myvector.assign(mydoubles,mydoubles+8);

  std::cout << "using 'compare_as_ints' :";
  std::stable_sort (myvector.begin(), myvector.end(), compare_as_ints);
  for (std::vector<double>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```

`compare_as_ints` is a function that compares only the integral part of the elements, therefore, elements with the same integral part are considered equivalent. `stable_sort` preserves the relative order these had before the call.

Possible output:

```
using default comparison: 1.32 1.41 1.62 1.73 2.58 2.72 3.14 4.67
using 'compare_as_ints' : 1.41 1.73 1.32 1.62 2.72 2.58 3.14 4.67
```

Complexity

If enough extra memory is available, linearithmic in the distance between *first* and *last*: Performs up to $N*\log_2(N)$ element comparisons (where *N* is this distance), and up to that many element moves.
Otherwise, polyloglinear in that distance: Performs up to $N*\log_2^2(N)$ element comparisons, and up to that many element swaps.

Data races

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

Exceptions

Throws if any of the element comparisons, the element swaps (or moves) or the operations on iterators throws.
Note that invalid arguments cause *undefined behavior*.

See also

sort	Sort elements in range (function template)
partial_sort	Partially sort elements in range (function template)
search	Search range for subsequence (function template)
reverse	Reverse range (function template)