skip_list

Generated by Doxygen 1.9.8

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 skip_list< T, Compare >::basic_iterator< IsConst > Class Template Reference

Iterator for skip_list.

`#include <skip_list.hpp>`

Public Types

- using iterator_category = std::forward_iterator_tag
- using value_type = T
- using difference_type = std::ptrdiff_t
- using reference = std::conditional_t< IsConst, const T &, T & >
- using pointer = std::conditional_t< IsConst, const T *, T * >

Public Member Functions

- basic_iterator ()
- template<bool B = IsConst, typename = std::enable_if_t<B>>
  basic_iterator (basic_iterator< false > const &other) noexcept
- basic_iterator & operator++ ()
- basic_iterator operator++ (int)
- reference operator* () const
- pointer operator-> () const
- bool operator== (basic_iterator const &o) const noexcept
- bool operator!= (basic_iterator const &o) const noexcept

Friends

- class skip_list

3.1.1 Detailed Description

template<typename T, typename Compare = std::less<T>>
template<bool IsConst>
class skip_list< T, Compare >::basic_iterator< IsConst >

Iterator for skip_list.

Template Parameters

IsConst	true for const_iterator, false for iterator.

Definition at line 64 of file skip_list.hpp.

3.1.2 Member Typedef Documentation

3.1.2.1 difference_type

```
template<typename T , typename Compare = std::less<T>>
template<bool IsConst>
using skip_list< T, Compare >::basic_iterator< IsConst >::difference_type = std::ptrdiff_t
```

Definition at line 74 of file skip_list.hpp.

3.1.2.2 iterator_category

```
template<typename T , typename Compare = std::less<T>>
template<bool IsConst>
using skip_list< T, Compare >::basic_iterator< IsConst >::iterator_category = std::forward_↩
iterator_tag
```

Definition at line 72 of file skip_list.hpp.

3.1.2.3 pointer

```
template<typename T , typename Compare = std::less<T>>
template<bool IsConst>
using skip_list< T, Compare >::basic_iterator< IsConst >::pointer = std::conditional_t<Is↩
Const, const T*, T*>
```

Definition at line 76 of file skip_list.hpp.

3.1.2.4 reference

```
template<typename T , typename Compare = std::less<T>>
template<bool IsConst>
using skip_list< T, Compare >::basic_iterator< IsConst >::reference = std::conditional_t<Is↩
Const, const T&, T&>
```

Definition at line 75 of file skip_list.hpp.

3.1.2.5 value_type

```
template<typename T , typename Compare = std::less<T>>
template<bool IsConst>
using skip_list< T, Compare >::basic_iterator< IsConst >::value_type = T
```

Definition at line 73 of file skip_list.hpp.

3.1.3 Constructor & Destructor Documentation

3.1.3.1 basic_iterator() [1/2]

```
template<typename T , typename Compare = std::less<T>>
template<bool IsConst>
skip_list< T, Compare >::basic_iterator< IsConst >::basic_iterator ( )  [inline]
```

Definition at line 78 of file skip_list.hpp.

```
00078 : ptr_(nullptr) {}
```

3.1.3.2 basic_iterator() [2/2]

```
template<typename T , typename Compare = std::less<T>>
template<bool IsConst>
template<bool B = IsConst, typename = std::enable_if_t<B>>
skip_list< T, Compare >::basic_iterator< IsConst >::basic_iterator (
            basic_iterator< false > const & other )  [inline], [noexcept]
```

Definition at line 81 of file skip_list.hpp.

```
00082                 : ptr_(other.ptr_) {}
```

3.1.4 Member Function Documentation

3.1.4.1 operator"!=()

```
template<typename T , typename Compare = std::less<T>>
template<bool IsConst>
bool skip_list< T, Compare >::basic_iterator< IsConst >::operator!= (
            basic_iterator< IsConst > const & o ) const  [inline], [noexcept]
```

Definition at line 108 of file skip_list.hpp.

```
00108                                                                 {
00109             return ptr_ != o.ptr_;
00110         }
```

3.1.4.2 operator*()

```
template<typename T , typename Compare = std::less<T>>
template<bool IsConst>
reference skip_list< T, Compare >::basic_iterator< IsConst >::operator* ( ) const  [inline]
```

Definition at line 95 of file skip_list.hpp.

```
00095                                 {
00096             if (!ptr_) throw std::out_of_range("SkipList iterator out of range");
00097             return ptr_->value;
00098         }
```

3.1.4.3 operator++() [1/2]

```
template<typename T , typename Compare = std::less<T>>
template<bool IsConst>
basic_iterator & skip_list< T, Compare >::basic_iterator< IsConst >::operator++ ( )  [inline]
```

Definition at line 84 of file skip_list.hpp.

```
00084                                 {
00085             if (ptr_) ptr_ = ptr_->forward[0];
00086             return *this;
00087         }
```

3.1.4.4 operator++() [2/2]

```
template<typename T , typename Compare = std::less<T>>
template<bool IsConst>
basic_iterator skip_list< T, Compare >::basic_iterator< IsConst >::operator++ (
            int  )  [inline]
```

Definition at line 89 of file skip_list.hpp.

```
00089                                       {
00090             auto tmp = *this;
00091             ++*this;
00092             return tmp;
00093         }
```

3.1.4.5 operator->()

```
template<typename T , typename Compare = std::less<T>>
template<bool IsConst>
pointer skip_list< T, Compare >::basic_iterator< IsConst >::operator-> ( ) const  [inline]
```

Definition at line 100 of file skip_list.hpp.

```
00100                                     {
00101             return std::addressof(operator*());
00102         }
```

3.1.4.6 operator==()

```
template<typename T , typename Compare = std::less<T>>
template<bool IsConst>
bool skip_list< T, Compare >::basic_iterator< IsConst >::operator== (
            basic_iterator< IsConst > const & o ) const  [inline], [noexcept]
```

Definition at line 104 of file skip_list.hpp.

```
00104                                                                {
00105             return ptr_ == o.ptr_;
00106         }
```

3.1.5 Friends And Related Symbol Documentation

3.1.5.1 skip_list

```
template<typename T , typename Compare = std::less<T>>
template<bool IsConst>
friend class skip_list  [friend]
```

Definition at line 65 of file skip_list.hpp.

The documentation for this class was generated from the following file:

- include/skip_list.hpp

3.2 skip_list< T, Compare > Class Template Reference

A skip list implementation for sorted storage and fast lookup.

`#include <skip_list.hpp>`

Classes

- class basic_iterator

  Iterator for skip_list.

Public Types

- using value_type = T
- using size_type = size_t
- using difference_type = std::ptrdiff_t
- using reference = T &
- using const_reference = const T &
- using key_type = T
- using key_compare = Compare
- using iterator = basic_iterator< false >
- using const_iterator = basic_iterator< true >

Public Member Functions

- skip_list ()

  Default constructor.
- skip_list (std::initializer_list< T > il)

  Construct from initializer list.
- template<typename InputIt , typename = std::enable_if_t<std::is_convertible_v< typename std::iterator_traits<InputIt>::iterator_category, std::input_iterator_tag>>>
  skip_list (InputIt first, InputIt last)

  Construct from input iterators.
- skip_list (const skip_list &other)

  Copy constructor.
- skip_list & operator= (const skip_list &other)

  Copy assignment.
- skip_list (skip_list &&o) noexcept

  Move constructor.
- skip_list & operator= (skip_list &&o) noexcept

  Move assignment.
- ~skip_list ()

  Destructor.
- bool empty () const noexcept

  Check if empty.
- size_type size () const noexcept

  Get number of elements.
- void clear () noexcept

  Remove all elements.
- std::pair< iterator, bool > insert (const T &value)

Insert a value.

- template<typename... Args>
  std::pair< iterator, bool > emplace (Args &&... args)

  Emplace a value (perfect-forwarded).
- size_type erase (const T &value)

  Remove value.
- iterator find (const T &value)

  Find element.
- const_iterator find (const T &value) const

  Find element.
- iterator lower_bound (const T &key)

  Get first element not less than key.
- const_iterator lower_bound (const T &key) const

  Get first element not less than key.
- iterator upper_bound (const T &key)

  Get first element greater than key.
- const_iterator upper_bound (const T &key) const

  Get first element greater than key.
- size_type count (const T &key) const

  Count occurrences of key.
- bool contains (const T &key) const

  Check if key exists.
- iterator begin () noexcept

  Iterator to first element.
- const_iterator begin () const noexcept

  Iterator to first element.
- const_iterator cbegin () const noexcept

  Const begin.
- iterator end () noexcept

  Iterator to end.
- const_iterator end () const noexcept

  Iterator to end.
- const_iterator cend () const noexcept

  Const end.
- bool operator== (skip_list const &rhs) const

  Equality comparison.
- bool operator!= (skip_list const &rhs) const

  Inequality comparison.

3.2.1 Detailed Description

template<typename T, typename Compare = std::less<T>>
class skip_list< T, Compare >

A skip list implementation for sorted storage and fast lookup.

Template Parameters

T	Type of elements stored.
Compare	Comparison functor (defaults to std::less<T>).

Definition at line 19 of file skip_list.hpp.

3.2.2 Member Typedef Documentation

3.2.2.1 const_iterator

```
template<typename T , typename Compare = std::less<T>>
using skip_list< T, Compare >::const_iterator = basic_iterator<true>
```

Definition at line 114 of file skip_list.hpp.

3.2.2.2 const_reference

```
template<typename T , typename Compare = std::less<T>>
using skip_list< T, Compare >::const_reference = const T&
```

Definition at line 55 of file skip_list.hpp.

3.2.2.3 difference_type

```
template<typename T , typename Compare = std::less<T>>
using skip_list< T, Compare >::difference_type = std::ptrdiff_t
```

Definition at line 53 of file skip_list.hpp.

3.2.2.4 iterator

```
template<typename T , typename Compare = std::less<T>>
using skip_list< T, Compare >::iterator = basic_iterator<false>
```

Definition at line 113 of file skip_list.hpp.

3.2.2.5 key_compare

```
template<typename T , typename Compare = std::less<T>>
using skip_list< T, Compare >::key_compare = Compare
```

Definition at line 57 of file skip_list.hpp.

3.2.2.6 key_type

```
template<typename T , typename Compare = std::less<T>>
using skip_list< T, Compare >::key_type = T
```

Definition at line 56 of file skip_list.hpp.

3.2.2.7 reference

```
template<typename T , typename Compare = std::less<T>>
using skip_list< T, Compare >::reference = T&
```

Definition at line 54 of file skip_list.hpp.

3.2.2.8 size_type

```
template<typename T , typename Compare = std::less<T>>
using skip_list< T, Compare >::size_type = size_t
```

Definition at line 52 of file skip_list.hpp.

3.2.2.9 value_type

```
template<typename T , typename Compare = std::less<T>>
using skip_list< T, Compare >::value_type = T
```

Definition at line 51 of file skip_list.hpp.

3.2.3 Constructor & Destructor Documentation

3.2.3.1 skip_list() [1/5]

```
template<typename T , typename Compare = std::less<T>>
skip_list< T, Compare >::skip_list ( )  [inline]
```

Default constructor.

Definition at line 117 of file skip_list.hpp.

```
00118             : head_(new Node(MAX_LEVEL, T{})),
00119               level_(0),
00120               size_(0),
00121               cmp_(Compare{}),
00122               gen_(std::random_device{}()),
00123               dist_(0.0, 1.0) {}
```

3.2.3.2 skip_list() [2/5]

```
template<typename T , typename Compare = std::less<T>>
skip_list< T, Compare >::skip_list (
            std::initializer_list< T > il )  [inline]
```

Construct from initializer list.

Parameters

il	List of values to insert.

Definition at line 129 of file skip_list.hpp.

```
00129                                       : skip_list() {
00130         for (auto const& v : il) insert(v);
00131     }
```

3.2.3.3 skip_list() [3/5]

```
template<typename T , typename Compare = std::less<T>>
template<typename InputIt , typename = std::enable_if_t<std::is_convertible_v< typename std↩
::iterator_traits<InputIt>::iterator_category, std::input_iterator_tag>>>
skip_list< T, Compare >::skip_list (
             InputIt first,
             InputIt last )  [inline]
```

Construct from input iterators.

Template Parameters

Input↩ It	Input iterator type.

Parameters

first	Begin iterator.
last	End iterator.

Definition at line 143 of file skip_list.hpp.

```
00143                                       : skip_list() {
00144         for (; first != last; ++first)
00145             insert(*first);
00146     }
```

3.2.3.4 skip_list() [4/5]

```
template<typename T , typename Compare = std::less<T>>
skip_list< T, Compare >::skip_list (
             const skip_list< T, Compare > & other )  [inline]
```

Copy constructor.

Definition at line 149 of file skip_list.hpp.

```
00149                                       : skip_list() {
00150         for (auto const& v : other)
00151             insert(v);
00152     }
```

3.2.3.5 skip_list() [5/5]

```
template<typename T , typename Compare = std::less<T>>
skip_list< T, Compare >::skip_list (
             skip_list< T, Compare > && o )  [inline], [noexcept]
```

Move constructor.

Definition at line 165 of file skip_list.hpp.

```
00166             : head_(o.head_),
00167               level_(o.level_),
00168       size_(o.size_),
00169       cmp_(std::move(o.cmp_)),
00170       gen_(std::random_device{}()),
00171       dist_(0.0, 1.0) {
00172           o.head_ = nullptr;
00173           o.size_ = 0;
00174       }
```

3.2.3.6 ~skip_list()

```
template<typename T , typename Compare = std::less<T>>
skip_list< T, Compare >::~skip_list ( )  [inline]
```

Destructor.

Definition at line 192 of file skip_list.hpp.

```
00192                   {
00193           if (head_) {
00194               clear();
00195               delete head_;
00196           }
00197       }
```

3.2.4 Member Function Documentation

3.2.4.1 begin() [1/2]

```
template<typename T , typename Compare = std::less<T>>
const_iterator skip_list< T, Compare >::begin ( ) const  [inline], [noexcept]
```

Iterator to first element.

Definition at line 374 of file skip_list.hpp.

```
00374                                           {
00375           return const_iterator(head_->forward[0]);
00376       }
```

3.2.4.2 begin() [2/2]

```
template<typename T , typename Compare = std::less<T>>
iterator skip_list< T, Compare >::begin ( )  [inline], [noexcept]
```

Iterator to first element.

Definition at line 369 of file skip_list.hpp.

```
00369                                     {
00370           return iterator(head_->forward[0]);
00371       }
```

3.2.4.3 cbegin()

```
template<typename T , typename Compare = std::less<T>>
const_iterator skip_list< T, Compare >::cbegin ( ) const  [inline], [noexcept]
```

Const begin.

Definition at line 379 of file skip_list.hpp.

```
00379                                           {
00380           return const_iterator(head_->forward[0]);
00381       }
```

3.2.4.4 cend()

```
template<typename T , typename Compare = std::less<T>>
const_iterator skip_list< T, Compare >::cend ( ) const  [inline], [noexcept]
```

Const end.

Definition at line 394 of file skip_list.hpp.

```
00394                                           {
00395         return const_iterator(nullptr);
00396     }
```

3.2.4.5 clear()

```
template<typename T , typename Compare = std::less<T>>
void skip_list< T, Compare >::clear ( )  [inline], [noexcept]
```

Remove all elements.

Definition at line 206 of file skip_list.hpp.

```
00206                                 {
00207         if (!head_) return;
00208
00209         Node* cur = head_->forward[0];
00210         while (cur) {
00211             Node* nxt = cur->forward[0];
00212             delete cur;
00213             cur = nxt;
00214         }
00215         std::fill(head_->forward.begin(), head_->forward.end(), nullptr);
00216         level_ = 0;
00217         size_  = 0;
00218     }
```

3.2.4.6 contains()

```
template<typename T , typename Compare = std::less<T>>
bool skip_list< T, Compare >::contains (
             const T & key ) const  [inline]
```

Check if key exists.

Parameters

key	Search key.

Returns

true if found.

Definition at line 364 of file skip_list.hpp.

```
00364                                          {
00365         return count(key) != 0;
00366     }
```

3.2.4.7 count()

```
template<typename T , typename Compare = std::less<T>>
size_type skip_list< T, Compare >::count (
            const T & key ) const  [inline]
```

Count occurrences of key.

Parameters

key	Search key.

Returns

1 if found, 0 otherwise.

Definition at line 355 of file skip_list.hpp.

```
00355                                           {
00356         return find(key) != end() ? 1 : 0;
00357     }
```

3.2.4.8 emplace()

```
template<typename T , typename Compare = std::less<T>>
template<typename...  Args>
std::pair< iterator, bool > skip_list< T, Compare >::emplace (
            Args &&...  args )  [inline]
```

Emplace a value (perfect-forwarded).

Parameters

args	Arguments to construct T.

Returns

Pair(iterator, true if inserted).

Definition at line 258 of file skip_list.hpp.

```
00258                                                   {
00259         T tmp(std::forward<Args>(args)...);
00260         return insert(tmp);
00261     }
```

3.2.4.9 empty()

```
template<typename T , typename Compare = std::less<T>>
bool skip_list< T, Compare >::empty ( ) const  [inline], [noexcept]
```

Check if empty.

Definition at line 200 of file skip_list.hpp.

```
00200 { return size_ == 0; }
```

3.2.4.10 end() [1/2]

```
template<typename T , typename Compare = std::less<T>>
const_iterator skip_list< T, Compare >::end ( ) const  [inline], [noexcept]
```

Iterator to end.

Definition at line 389 of file skip_list.hpp.

```
00389                                       {
00390         return const_iterator(nullptr);
00391     }
```

3.2.4.11 end() [2/2]

```
template<typename T , typename Compare = std::less<T>>
iterator skip_list< T, Compare >::end ( )  [inline], [noexcept]
```

Iterator to end.

Definition at line 384 of file skip_list.hpp.

```
00384                                 {
00385         return iterator(nullptr);
00386     }
```

3.2.4.12 erase()

```
template<typename T , typename Compare = std::less<T>>
size_type skip_list< T, Compare >::erase (
             const T & value )  [inline]
```

Remove value.

Parameters

value	Value to erase.

Returns

Number of elements removed (0 or 1).

Definition at line 268 of file skip_list.hpp.

```
00268                                       {
00269         std::vector<Node*> update(MAX_LEVEL + 1);
00270         Node* x = head_;
00271         for (int i = level_; i >= 0; --i) {
00272             while (x->forward[i] && cmp_(x->forward[i]->value, value))
00273                 x = x->forward[i];
00274             update[i] = x;
00275         }
00276         x = x->forward[0];
00277         if (!x || cmp_(value, x->value) || cmp_(x->value, value))
00278             return 0;
00279         for (int i = 0; i <= level_; ++i) {
00280             if (update[i]->forward[i] != x) break;
00281             update[i]->forward[i] = x->forward[i];
00282         }
00283         delete x;
00284         while (level_ > 0 && head_->forward[level_] == nullptr)
00285             --level_;
00286         --size_;
00287         return 1;
00288     }
```

3.2.4.13 find() [1/2]

```
template<typename T , typename Compare = std::less<T>>
iterator skip_list< T, Compare >::find (
            const T & value )  [inline]
```

Find element.

Parameters

value	Key to find.

Returns

Iterator to element or end().

Definition at line 295 of file skip_list.hpp.

```
00295                                    {
00296         Node* x = head_;
00297         for (int i = level_; i >= 0; --i) {
00298             while (x->forward[i] && cmp_(x->forward[i]->value, value))
00299                 x = x->forward[i];
00300         }
00301         x = x->forward[0];
00302         if (x && !cmp_(value, x->value) && !cmp_(x->value, value))
00303             return iterator(x);
00304         return end();
00305     }
```

3.2.4.14 find() [2/2]

```
template<typename T , typename Compare = std::less<T>>
const_iterator skip_list< T, Compare >::find (
            const T & value ) const  [inline]
```

Find element.

Parameters

value	Key to find.

Returns

Iterator to element or end().

Definition at line 308 of file skip_list.hpp.

```
00308                                          {
00309         return const_cast<skip_list*>(this)->find(value);
00310     }
```

3.2.4.15 insert()

```
template<typename T , typename Compare = std::less<T>>
std::pair< iterator, bool > skip_list< T, Compare >::insert (
            const T & value )  [inline]
```

Insert a value.

Parameters

value	Value to insert.

Returns

Pair(iterator to new or existing element, true if inserted).

Definition at line 225 of file skip_list.hpp.

```
00225                                                 {
00226         std::vector<Node*> update(MAX_LEVEL + 1);
00227         Node* x = head_;
00228         for (int i = level_; i >= 0; --i) {
00229             while (x->forward[i] && cmp_(x->forward[i]->value, value))
00230                 x = x->forward[i];
00231             update[i] = x;
00232         }
00233         x = x->forward[0];
00234         if (x && !cmp_(value, x->value) && !cmp_(x->value, value)) {
00235             return {iterator(x), false};
00236         }
00237         int lvl = random_level();
00238         if (lvl > level_) {
00239             for (int i = level_ + 1; i <= lvl; ++i)
00240                 update[i] = head_;
00241             level_ = lvl;
00242         }
00243         Node* newNode = new Node(lvl, value);
00244         for (int i = 0; i <= lvl; ++i) {
00245             newNode->forward[i] = update[i]->forward[i];
00246             update[i]->forward[i] = newNode;
00247         }
00248         ++size_;
00249         return {iterator(newNode), true};
00250     }
```

3.2.4.16 lower_bound() [1/2]

```
template<typename T , typename Compare = std::less<T>>
iterator skip_list< T, Compare >::lower_bound (
            const T & key )  [inline]
```

Get first element not less than key.

Parameters

key	Search key.

Returns

Lower bound iterator.

Definition at line 317 of file skip_list.hpp.

```
00317                                           {
00318         Node* x = head_;
00319         for (int i = level_; i >= 0; --i) {
00320             while (x->forward[i] && cmp_(x->forward[i]->value, key))
00321                 x = x->forward[i];
00322         }
00323         return iterator(x->forward[0]);
00324     }
```

3.2.4.17 lower_bound() [2/2]

```
template<typename T , typename Compare = std::less<T>>
const_iterator skip_list< T, Compare >::lower_bound (
            const T & key ) const  [inline]
```

Get first element not less than key.

Parameters

key	Search key.

Returns

Lower bound iterator.

Definition at line 327 of file skip_list.hpp.

```
00327                                                   {
00328         return const_cast<skip_list*>(this)->lower_bound(key);
00329     }
```

3.2.4.18 operator"!=()

```
template<typename T , typename Compare = std::less<T>>
bool skip_list< T, Compare >::operator!= (
            skip_list< T, Compare > const & rhs ) const  [inline]
```

Inequality comparison.

Definition at line 415 of file skip_list.hpp.

```
00415                                                {
00416         return !(*this == rhs);
00417     }
```

3.2.4.19 operator=() [1/2]

```
template<typename T , typename Compare = std::less<T>>
skip_list & skip_list< T, Compare >::operator= (
            const skip_list< T, Compare > & other )  [inline]
```

Copy assignment.

Definition at line 155 of file skip_list.hpp.

```
00155                                                 {
00156         if (this != &other) {
00157             clear();
00158             for (auto const& v : other)
00159                 insert(v);
00160         }
00161         return *this;
00162     }
```

3.2.4.20 operator=() [2/2]

```
template<typename T , typename Compare = std::less<T>>
skip_list & skip_list< T, Compare >::operator= (
            skip_list< T, Compare > && o )  [inline], [noexcept]
```

Move assignment.

Definition at line 177 of file skip_list.hpp.

```
00177                                                   {
00178         if (this != &o) {
00179             clear();
00180             delete head_;
00181             head_    = o.head_;
00182             level_   = o.level_;
00183             size_    = o.size_;
00184             cmp_     = std::move(o.cmp_);
00185             o.head_  = nullptr;
00186             o.size_  = 0;
00187         }
00188         return *this;
00189     }
```

3.2.4.21 operator==()

```
template<typename T , typename Compare = std::less<T>>
bool skip_list< T, Compare >::operator== (
            skip_list< T, Compare > const & rhs ) const  [inline]
```

Equality comparison.

Parameters

rhs	Other skip_list.

Returns

true if sizes and elements match.

Definition at line 403 of file skip_list.hpp.

```
00403                                                   {
00404         if (size_ != rhs.size_) return false;
00405         auto it1 = begin();
00406         auto it2 = rhs.begin();
00407         while (it1 != end()) {
00408             if (*it1 != *it2) return false;
00409             ++it1; ++it2;
00410         }
00411         return true;
00412     }
```

3.2.4.22 size()

```
template<typename T , typename Compare = std::less<T>>
size_type skip_list< T, Compare >::size ( ) const  [inline], [noexcept]
```

Get number of elements.

Definition at line 203 of file skip_list.hpp.

```
00203 { return size_; }
```

3.2.4.23 upper_bound() [1/2]

```
template<typename T , typename Compare = std::less<T>>
iterator skip_list< T, Compare >::upper_bound (
            const T & key )  [inline]
```

Get first element greater than key.

Parameters

key	Search key.

Returns

Upper bound iterator.

Definition at line 336 of file skip_list.hpp.

```
00336                                       {
00337         Node* x = head_;
00338         for (int i = level_; i >= 0; --i) {
00339             while (x->forward[i] && !cmp_(key, x->forward[i]->value))
00340                 x = x->forward[i];
00341         }
00342         return iterator(x->forward[0]);
00343     }
```

3.2.4.24 upper_bound() [2/2]

```
template<typename T , typename Compare = std::less<T>>
const_iterator skip_list< T, Compare >::upper_bound (
            const T & key ) const  [inline]
```

Get first element greater than key.

Parameters

key	Search key.

Returns

Upper bound iterator.

Definition at line 346 of file skip_list.hpp.

```
00346                                                 {
00347         return const_cast<skip_list*>(this)->upper_bound(key);
00348     }
```

The documentation for this class was generated from the following file:

- include/skip_list.hpp

Chapter 4

File Documentation

4.1 include/skip_list.hpp File Reference

```
#include <algorithm>
#include <functional>
#include <iterator>
#include <limits>
#include <memory>
#include <random>
#include <stdexcept>
#include <type_traits>
#include <vector>
```

Include dependency graph for skip_list.hpp:

4.2 skip_list.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002
00003 #include <algorithm>
00004 #include <functional>
00005 #include <iterator>
00006 #include <limits>
00007 #include <memory>
00008 #include <random>
00009 #include <stdexcept>
00010 #include <type_traits>
00011 #include <vector>
00012
00018 template <typename T, typename Compare = std::less<T»
00019 class skip_list {
00020 private:
00021     struct Node {
00022         T value;
00023         std::vector<Node*> forward;
00024
00025         Node(int level, const T& val)
00026                 : value(val), forward(level + 1, nullptr) {}
00027     };
00028
00029     static constexpr int MAX_LEVEL = 16;
00030     static constexpr double P = 0.5;
00031
00032     Node* head_;
00033     int level_;
00034     size_t size_;
00035     Compare cmp_;
00036     std::mt19937 gen_;
```

```
00037     std::uniform_real_distribution<> dist_;
00038
00043     int random_level() {
00044         int lvl = 0;
00045         while (lvl < MAX_LEVEL && dist_(gen_) < P)
00046             ++lvl;
00047         return lvl;
00048     }
00049
00050 public:
00051     using value_type        = T;
00052     using size_type         = size_t;
00053     using difference_type   = std::ptrdiff_t;
00054     using reference         = T&;
00055     using const_reference   = const T&;
00056     using key_type          = T;
00057     using key_compare       = Compare;
00058
00063     template <bool IsConst>
00064     class basic_iterator {
00065         friend class skip_list;
00066         using node_ptr = std::conditional_t<IsConst, const Node*, Node*>;
00067         node_ptr ptr_;
00068
00069         explicit basic_iterator(node_ptr p) : ptr_(p) {}
00070
00071     public:
00072         using iterator_category = std::forward_iterator_tag;
00073         using value_type        = T;
00074         using difference_type   = std::ptrdiff_t;
00075         using reference         = std::conditional_t<IsConst, const T&, T&>;
00076         using pointer           = std::conditional_t<IsConst, const T*, T*>;
00077
00078         basic_iterator() : ptr_(nullptr) {}
00079
00080         template <bool B = IsConst, typename = std::enable_if_t<B»
00081         basic_iterator(basic_iterator<false> const& other) noexcept
00082                 : ptr_(other.ptr_) {}
00083
00084         basic_iterator& operator++() {
00085             if (ptr_) ptr_ = ptr_->forward[0];
00086             return *this;
00087         }
00088
00089         basic_iterator operator++(int) {
00090             auto tmp = *this;
00091             ++*this;
00092             return tmp;
00093         }
00094
00095         reference operator*() const {
00096             if (!ptr_) throw std::out_of_range("SkipList iterator out of range");
00097             return ptr_->value;
00098         }
00099
00100         pointer operator->() const {
00101             return std::addressof(operator*());
00102         }
00103
00104         bool operator==(basic_iterator const& o) const noexcept {
00105             return ptr_ == o.ptr_;
00106         }
00107
00108         bool operator!=(basic_iterator const& o) const noexcept {
00109             return ptr_ != o.ptr_;
00110         }
00111     };
00112
00113     using iterator       = basic_iterator<false>;
00114     using const_iterator = basic_iterator<true>;
00115
00117     skip_list()
00118             : head_(new Node(MAX_LEVEL, T{})),
00119               level_(0),
00120               size_(0),
00121               cmp_(Compare{}),
00122               gen_(std::random_device{}()),
00123               dist_(0.0, 1.0) {}
00124
00129     skip_list(std::initializer_list<T> il) : skip_list() {
00130         for (auto const& v : il) insert(v);
00131     }
00132
00139     template <typename InputIt,
00140             typename = std::enable_if_t<std::is_convertible_v<
00141                     typename std::iterator_traits<InputIt>::iterator_category,
00142                     std::input_iterator_tag»>
```

```
00143     skip_list(InputIt first, InputIt last) : skip_list() {
00144         for (; first != last; ++first)
00145             insert(*first);
00146     }
00147
00149     skip_list(const skip_list& other) : skip_list() {
00150         for (auto const& v : other)
00151             insert(v);
00152     }
00153
00155     skip_list& operator=(const skip_list& other) {
00156         if (this != &other) {
00157             clear();
00158             for (auto const& v : other)
00159                 insert(v);
00160         }
00161         return *this;
00162     }
00163
00165     skip_list(skip_list&& o) noexcept
00166             : head_(o.head_),
00167             level_(o.level_),
00168     size_(o.size_),
00169     cmp_(std::move(o.cmp_)),
00170     gen_(std::random_device{}()),
00171     dist_(0.0, 1.0) {
00172         o.head_ = nullptr;
00173         o.size_ = 0;
00174     }
00175
00177     skip_list& operator=(skip_list&& o) noexcept {
00178         if (this != &o) {
00179             clear();
00180             delete head_;
00181             head_    = o.head_;
00182             level_   = o.level_;
00183             size_    = o.size_;
00184             cmp_     = std::move(o.cmp_);
00185             o.head_  = nullptr;
00186             o.size_  = 0;
00187         }
00188         return *this;
00189     }
00190
00192     ~skip_list() {
00193         if (head_) {
00194             clear();
00195             delete head_;
00196         }
00197     }
00198
00200     bool empty() const noexcept { return size_ == 0; }
00201
00203     size_type size() const noexcept { return size_; }
00204
00206     void clear() noexcept {
00207         if (!head_) return;
00208
00209         Node* cur = head_->forward[0];
00210         while (cur) {
00211             Node* nxt = cur->forward[0];
00212             delete cur;
00213             cur = nxt;
00214         }
00215         std::fill(head_->forward.begin(), head_->forward.end(), nullptr);
00216         level_ = 0;
00217         size_  = 0;
00218     }
00219
00225     std::pair<iterator, bool> insert(const T& value) {
00226         std::vector<Node*> update(MAX_LEVEL + 1);
00227         Node* x = head_;
00228         for (int i = level_; i >= 0; --i) {
00229             while (x->forward[i] && cmp_(x->forward[i]->value, value))
00230                 x = x->forward[i];
00231             update[i] = x;
00232         }
00233         x = x->forward[0];
00234         if (x && !cmp_(value, x->value) && !cmp_(x->value, value)) {
00235             return {iterator(x), false};
00236         }
00237         int lvl = random_level();
00238         if (lvl > level_) {
00239             for (int i = level_ + 1; i <= lvl; ++i)
00240                 update[i] = head_;
00241             level_ = lvl;
00242         }
```

```
00243         Node* newNode = new Node(lvl, value);
00244         for (int i = 0; i <= lvl; ++i) {
00245             newNode->forward[i] = update[i]->forward[i];
00246             update[i]->forward[i] = newNode;
00247         }
00248         ++size_;
00249         return {iterator(newNode), true};
00250     }
00251
00257     template <typename... Args>
00258     std::pair<iterator, bool> emplace(Args&&... args) {
00259         T tmp(std::forward<Args>(args)...);
00260         return insert(tmp);
00261     }
00262
00268     size_type erase(const T& value) {
00269         std::vector<Node*> update(MAX_LEVEL + 1);
00270         Node* x = head_;
00271         for (int i = level_; i >= 0; --i) {
00272             while (x->forward[i] && cmp_(x->forward[i]->value, value))
00273                 x = x->forward[i];
00274             update[i] = x;
00275         }
00276         x = x->forward[0];
00277         if (!x || cmp_(value, x->value) || cmp_(x->value, value))
00278             return 0;
00279         for (int i = 0; i <= level_; ++i) {
00280             if (update[i]->forward[i] != x) break;
00281             update[i]->forward[i] = x->forward[i];
00282         }
00283         delete x;
00284         while (level_ > 0 && head_->forward[level_] == nullptr)
00285             --level_;
00286         --size_;
00287         return 1;
00288     }
00289
00295     iterator find(const T& value) {
00296         Node* x = head_;
00297         for (int i = level_; i >= 0; --i) {
00298             while (x->forward[i] && cmp_(x->forward[i]->value, value))
00299                 x = x->forward[i];
00300         }
00301         x = x->forward[0];
00302         if (x && !cmp_(value, x->value) && !cmp_(x->value, value))
00303             return iterator(x);
00304         return end();
00305     }
00306
00308     const_iterator find(const T& value) const {
00309         return const_cast<skip_list*>(this)->find(value);
00310     }
00311
00317     iterator lower_bound(const T& key) {
00318         Node* x = head_;
00319         for (int i = level_; i >= 0; --i) {
00320             while (x->forward[i] && cmp_(x->forward[i]->value, key))
00321                 x = x->forward[i];
00322         }
00323         return iterator(x->forward[0]);
00324     }
00325
00327     const_iterator lower_bound(const T& key) const {
00328         return const_cast<skip_list*>(this)->lower_bound(key);
00329     }
00330
00336     iterator upper_bound(const T& key) {
00337         Node* x = head_;
00338         for (int i = level_; i >= 0; --i) {
00339             while (x->forward[i] && !cmp_(key, x->forward[i]->value))
00340                 x = x->forward[i];
00341         }
00342         return iterator(x->forward[0]);
00343     }
00344
00346     const_iterator upper_bound(const T& key) const {
00347         return const_cast<skip_list*>(this)->upper_bound(key);
00348     }
00349
00355     size_type count(const T& key) const {
00356         return find(key) != end() ? 1 : 0;
00357     }
00358
00364     bool contains(const T& key) const {
00365         return count(key) != 0;
00366     }
00367
```

```
00369     iterator begin() noexcept {
00370         return iterator(head_->forward[0]);
00371     }
00372
00374     const_iterator begin() const noexcept {
00375         return const_iterator(head_->forward[0]);
00376     }
00377
00379     const_iterator cbegin() const noexcept {
00380         return const_iterator(head_->forward[0]);
00381     }
00382
00384     iterator end() noexcept {
00385         return iterator(nullptr);
00386     }
00387
00389     const_iterator end() const noexcept {
00390         return const_iterator(nullptr);
00391     }
00392
00394     const_iterator cend() const noexcept {
00395         return const_iterator(nullptr);
00396     }
00397
00403     bool operator==(skip_list const& rhs) const {
00404         if (size_ != rhs.size_) return false;
00405         auto it1 = begin();
00406         auto it2 = rhs.begin();
00407         while (it1 != end()) {
00408             if (*it1 != *it2) return false;
00409             ++it1; ++it2;
00410         }
00411         return true;
00412     }
00413
00415     bool operator!=(skip_list const& rhs) const {
00416         return !(*this == rhs);
00417     }
00418 };
```

Index