Data Structures Library

Generated by Doxygen 1.8.13

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

Class Documentation

2.1 dsl::hashmap< key, value, hash, equal > Class Template Reference

```
#include <hashmap.h>
```

Classes

- struct iterator

Public Member Functions

- **hashmap** (size_t bucket_count)
- iterator begin ()
- iterator end ()
- iterator find (key id)
- void insert (const std::pair< key, value > &element)
- void erase (iterator it)
- size_t size () const
- bool empty () const
- void clear ()

2.1.1 Detailed Description

template<class key, class value, class hash = std::hash<key>, class equal = std::equal_to<key>>
class dsl::hashmap< key, value, hash, equal >

This is an implementation of a hashmap that uses linear probing to solve collisions.

It uses buckets of std::vector to store values.

Template Parameters

key	The type of the key value of an entry.
value	The type of the mapped value of an entry.
hash	A unary function object, used to retrieve the hash code of a key to order elements into buckets.
equal	A binary predicate, used to compare two keys for equality.

2.1.2 Member Function Documentation

2.1.2.1 begin()

```
template<class key , class value , class hash = std::hash<key>, class equal = std::equal_↩
to<key>>
iterator dsl::hashmap< key, value, hash, equal >::begin ( )  [inline]
```

Finds the first element of the map, by iterating through the buckets, until a non-empty bucket is found.

2.1.2.2 clear()

```
template<class key , class value , class hash = std::hash<key>, class equal = std::equal_↩
to<key>>
void dsl::hashmap< key, value, hash, equal >::clear ( )  [inline]
```

Clears the hashmap, by removing all the elements from all the buckets.

2.1.2.3 empty()

```
template<class key , class value , class hash = std::hash<key>, class equal = std::equal_↩
to<key>>
bool dsl::hashmap< key, value, hash, equal >::empty ( ) const  [inline]
```

Checks if the hashmap is empty.

2.1.2.4 end()

```
template<class key , class value , class hash = std::hash<key>, class equal = std::equal_↩
to<key>>
iterator dsl::hashmap< key, value, hash, equal >::end ( )  [inline]
```

Returns an iterator to the end of the map.

2.1.2.5 erase()

```
template<class key , class value , class hash = std::hash<key>, class equal = std::equal_↩
to<key>>
void dsl::hashmap< key, value, hash, equal >::erase (
             iterator it )  [inline]
```

Erases the element at the given iterator. If the iterator is not valid, the behaviour is undefined.

2.1.2.6 find()

```
template<class key , class value , class hash = std::hash<key>, class equal = std::equal_↩
to<key>>
iterator dsl::hashmap< key, value, hash, equal >::find (
             key id )  [inline]
```

Returns an iterator to the element identified by the key. If no element has the given key, return the end iterator.

2.1.2.7 insert()

```
template<class key , class value , class hash = std::hash<key>, class equal = std::equal_↩
to<key>>
void dsl::hashmap< key, value, hash, equal >::insert (
             const std::pair< key, value > & element )  [inline]
```

Inserts a new element into the hashmap. If the key is already in the hashmap, don't modify its value.

2.1.2.8 size()

```
template<class key , class value , class hash = std::hash<key>, class equal = std::equal_↩
to<key>>
size_t dsl::hashmap< key, value, hash, equal >::size ( ) const  [inline]
```

Returns the number of elements in the hashmap.

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

- include/dsl/hashmap.h

2.2 dsl::heap< type, compare > Class Template Reference

```
#include <heap.h>
```

Public Member Functions

- template<class Iter >
  heap (Iter first, Iter last)
- size_t size () const
- bool empty () const
- void push (type value)
- void pop ()
- type top () const
- void clear ()

2.2.1 Detailed Description

template<class type, class compare = std::less<type>>
class dsl::heap< type, compare >

This is an implementation of a priority queue, using a heap structure.

It uses the std::vector container to hold the elements.

Template Parameters

type	The type of the value of an entry in the heap.
compare	A binary predicate that defines a strict weak ordering, used to order the elements. The expression compare(a,b) shall return true if a is considered to go before b.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 heap()

```
template<class type , class compare = std::less<type>>
template<class Iter >
dsl::heap< type, compare >::heap (
             Iter first,
             Iter last )  [inline]
```

Constructs the heap by inserting the elements in the range [first,last) and then sorting the heap.

2.2.3 Member Function Documentation

2.2.3.1 clear()

```
template<class type , class compare = std::less<type>>
void dsl::heap< type, compare >::clear ( )  [inline]
```

Removes all elements from the heap.

2.2.3.2 empty()

```
template<class type , class compare = std::less<type>>
bool dsl::heap< type, compare >::empty ( ) const  [inline]
```

Checks if the heap is emtpy.

2.2.3.3 pop()

```
template<class type , class compare = std::less<type>>
void dsl::heap< type, compare >::pop ( )  [inline]
```

Removes the top element from the heap.

2.2.3.4 push()

```
template<class type , class compare = std::less<type>>
void dsl::heap< type, compare >::push (
             type value )  [inline]
```

Inserts a new value into the heap.

2.2.3.5 size()

```
template<class type , class compare = std::less<type>>
size_t dsl::heap< type, compare >::size ( ) const  [inline]
```

Returns the number of elements in the heap.

2.2.3.6 top()

```
template<class type , class compare = std::less<type>>
type dsl::heap< type, compare >::top ( ) const  [inline]
```

Returns the value of the top element of the heap.

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

- include/dsl/heap.h

2.3 dsl::set< key, compare >::iterator Struct Reference

```
#include <set.h>
```

Public Types

- using **iterator_category** = std::bidirectional_iterator_tag
- using **difference_type** = std::ptrdiff_t
- using **value_type** = const key
- using **pointer** = const key ∗
- using **reference** = const key &

Public Member Functions

- **iterator** (node ∗here, tree ∗structure)
- reference operator∗ () const
- pointer operator-> ()
- iterator & operator++ ()
- iterator operator++ (int)
- iterator & operator-- ()
- iterator operator-- (int)

Friends

- class **set**
- bool operator== (const iterator &a, const iterator &b)
- bool operator!= (const iterator &a, const iterator &b)

2.3.1 Detailed Description

template<class key, class compare = std::less<key>>
struct dsl::set< key, compare >::iterator

This is the iterator for the set. Iterating through the set returns the elements in the order defined by the compare predicate

2.3.2 Member Function Documentation

2.3.2.1 operator*()

```
template<class key , class compare = std::less<key>>
reference dsl::set< key, compare >::iterator::operator* ( ) const  [inline]
```

De-references the iterator.

2.3.2.2 operator++() [1/2]

```
template<class key , class compare = std::less<key>>
iterator& dsl::set< key, compare >::iterator::operator++ ( )  [inline]
```

Incrementing this iterator is finding the successor of the element the iterator points at.

2.3.2.3 operator++() [2/2]

```
template<class key , class compare = std::less<key>>
iterator dsl::set< key, compare >::iterator::operator++ (
             int  )  [inline]
```

Post-increment, same as pre-increment, but return the value before the increment.

2.3.2.4 operator--() [1/2]

```
template<class key , class compare = std::less<key>>
iterator& dsl::set< key, compare >::iterator::operator-- ( )  [inline]
```

Decrementing this iterator is finding the successor of the element the iterator points at.

2.3.2.5 operator--() [2/2]

```
template<class key , class compare = std::less<key>>
iterator dsl::set< key, compare >::iterator::operator-- (
             int  )  [inline]
```

Post-decrement, same as pre-decrement, but return the value before the decrement.

2.3.2.6 operator->()

```
template<class key , class compare = std::less<key>>
pointer dsl::set< key, compare >::iterator::operator-> ( )  [inline]
```

De-references the iterator.

2.3.3 Friends And Related Function Documentation

2.3.3.1 operator"!=

```
template<class key , class compare = std::less<key>>
bool operator!= (
             const iterator & a,
             const iterator & b )  [friend]
```

Checks if two iterators are not equal.

2.3.3.2 operator==

```
template<class key , class compare = std::less<key>>
bool operator== (
             const iterator & a,
             const iterator & b )  [friend]
```

Checks if two iterators are equal.

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

- include/dsl/set.h

2.4 dsl::hashmap< key, value, hash, equal >::iterator Struct Reference

```
#include <hashmap.h>
```

Public Types

- using **iterator_category** = std::forward_iterator_tag
- using **difference_type** = std::ptrdiff_t
- using **value_type** = std::pair< key, value >
- using **pointer** = std::pair< key, value > ∗
- using **reference** = std::pair< key, value > &

Public Member Functions

- **iterator** (node here)
- reference operator∗ () const
- pointer operator-> ()
- iterator & operator++ ()
- iterator operator++ (int)

Friends

- class **hashmap**
- bool operator== (const iterator &a, const iterator &b)
- bool operator!= (const iterator &a, const iterator &b)

2.4.1 Detailed Description

template<class key, class value, class hash = std::hash<key>, class equal = std::equal_to<key>>
struct dsl::hashmap< key, value, hash, equal >::iterator

This is the iterator for the hashmap. Iterating through the map returns the elements in a seemingly random order.

2.4.2 Member Function Documentation

2.4.2.1 operator∗()

```
template<class key , class value , class hash = std::hash<key>, class equal = std::equal_←
to<key>>
reference dsl::hashmap< key, value, hash, equal >::iterator::operator∗ ( ) const  [inline]
```

De-references the iterator. The key should not be modified.

2.4.2.2 operator++() [1/2]

```
template<class key , class value , class hash = std::hash<key>, class equal = std::equal_←
to<key>>
iterator& dsl::hashmap< key, value, hash, equal >::iterator::operator++ ( )  [inline]
```

Incrementing this iterator is finding the next value, skipping empty buckets

2.4.2.3 operator++() [2/2]

```
template<class key , class value , class hash = std::hash<key>, class equal = std::equal_↩
to<key>>
iterator dsl::hashmap< key, value, hash, equal >::iterator::operator++ (
             int  )  [inline]
```

Post-increment, same as pre-increment, but return the value before the increment

2.4.2.4 operator->()

```
template<class key , class value , class hash = std::hash<key>, class equal = std::equal_↩
to<key>>
pointer dsl::hashmap< key, value, hash, equal >::iterator::operator-> ( )  [inline]
```

De-references the iterator. The key should not be modified.

2.4.3 Friends And Related Function Documentation

2.4.3.1 operator"!=

```
template<class key , class value , class hash = std::hash<key>, class equal = std::equal_↩
to<key>>
bool operator!= (
             const iterator & a,
             const iterator & b )  [friend]
```

Checks if two iterators are not equal.

2.4.3.2 operator==

```
template<class key , class value , class hash = std::hash<key>, class equal = std::equal_↩
to<key>>
bool operator== (
             const iterator & a,
             const iterator & b )  [friend]
```

Checks if two iterators are equal.

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

- include/dsl/hashmap.h

2.5 dsl::list< type >::iterator Struct Reference

```
#include <list.h>
```

Public Types

- using **iterator_category** = std::bidirectional_iterator_tag
- using **difference_type** = std::ptrdiff_t
- using **value_type** = type
- using **pointer** = type ∗
- using **reference** = type &

Public Member Functions

- **iterator** (node ∗position)
- reference operator∗ () const
- pointer operator-> ()
- iterator & operator++ ()
- iterator operator++ (int)
- iterator & operator-- ()
- iterator operator-- (int)

Friends

- class **list**
- bool operator== (const iterator &a, const iterator &b)
- bool operator!= (const iterator &a, const iterator &b)

2.5.1 Detailed Description

template<class type>
struct dsl::list< type >::iterator

This is the iterator for the list. Iterating through the list returns elements in the order they were inserted.

2.5.2 Member Function Documentation

2.5.2.1 operator∗()

```
template<class type >
reference dsl::list< type >::iterator::operator∗ ( ) const  [inline]
```

De-references the iterator.

2.5.2.2 operator++() [1/2]

```
template<class type >
iterator& dsl::list< type >::iterator::operator++ ( )  [inline]
```

Prefix increment, just move to the next element in the list.

2.5.2.3 operator++() [2/2]

```
template<class type >
iterator dsl::list< type >::iterator::operator++ (
             int  )  [inline]
```

Same as prefix increment, but return the value before the increment.

2.5.2.4 operator--() [1/2]

```
template<class type >
iterator& dsl::list< type >::iterator::operator-- ( )  [inline]
```

Prefix decrement, just move to the previous element in the list.

2.5.2.5 operator--() [2/2]

```
template<class type >
iterator dsl::list< type >::iterator::operator-- (
             int  )  [inline]
```

Same as prefix decrement, but return the value before the decrement.

2.5.2.6 operator->()

```
template<class type >
pointer dsl::list< type >::iterator::operator-> ( )  [inline]
```

De-references the iterator.

2.5.3 Friends And Related Function Documentation

2.5.3.1 operator"!=

```
template<class type >
bool operator!= (
             const iterator & a,
             const iterator & b )  [friend]
```

Checks if two iterators are not equal.

2.5.3.2 operator==

```
template<class type >
bool operator== (
             const iterator & a,
             const iterator & b )  [friend]
```

Checks if two iterators are equal.

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

- include/dsl/list.h

2.6 dsl::list< type > Class Template Reference

```
#include <list.h>
```

Classes

- struct iterator

Public Member Functions

- list (const list &other)
- list & operator= (list other)
- list (list &&other) noexcept
- void swap (list &other)
- ∼list ()
- iterator begin ()
- iterator end ()
- iterator insert (iterator position, const type &value)
- iterator erase (iterator position)
- size_t size () const
- bool empty () const
- void clear ()
- type & front ()
- type & back ()

2.6.1 Detailed Description

template<class type>
class dsl::list< type >

This class is an implementation of a doubly linked list.

Template Parameters

type	The type of a value of an entry in the list.

2.6.2 Constructor & Destructor Documentation

2.6.2.1 list() [1/2]

```
template<class type >
dsl::list< type >::list (
             const list< type > & other )  [inline]
```

Copy constructor, make a copy of the other list.

2.6.2.2 list() [2/2]

```
template<class type >
dsl::list< type >::list (
             list< type > && other )  [inline], [noexcept]
```

Swaps the content of this list with the content of the rvalue list.

2.6.2.3 ∼list()

```
template<class type >
dsl::list< type >::∼list ( )  [inline]
```

Destroys the list object.

2.6.3 Member Function Documentation

2.6.3.1 back()

```
template<class type >
type& dsl::list< type >::back ( )  [inline]
```

Returns a reference to the last element.

Calling this function when the list is empty results in undefined behaviour.

2.6.3.2 begin()

```
template<class type >
iterator dsl::list< type >::begin ( )  [inline]
```

Returns an iterator that points to the beginning of the list.

2.6.3.3 clear()

```
template<class type >
void dsl::list< type >::clear ( )  [inline]
```

Removes all the elements from the list.

2.6.3.4 empty()

```
template<class type >
bool dsl::list< type >::empty ( ) const  [inline]
```

Checks if the list is empty.

2.6.3.5 end()

```
template<class type >
iterator dsl::list< type >::end ( )  [inline]
```

Returns an iterator that points to the end of the list.

2.6.3.6 erase()

```
template<class type >
iterator dsl::list< type >::erase (
             iterator position )  [inline]
```

Erases the element at the specified position.

It returns an iterator to the element that followed the erased element.

2.6.3.7 front()

```
template<class type >
type& dsl::list< type >::front ( )  [inline]
```

Returns a reference to the first element.

Calling this function when the list is empty results in undefined behaviour.

2.6.3.8 insert()

```
template<class type >
iterator dsl::list< type >::insert (
             iterator position,
             const type & value )  [inline]
```

Inserts the given value before the element at the specified position.

It returns an iterator to the newly inserted element.

2.6.3.9 operator=()

```
template<class type >
list& dsl::list< type >::operator= (
             list< type > other )  [inline]
```

Assigns new contents to the list, replacing its current contents.

2.6.3.10 size()

```
template<class type >
size_t dsl::list< type >::size ( ) const  [inline]
```

Returns the number of elements in the list.

2.6.3.11 swap()

```
template<class type >
void dsl::list< type >::swap (
             list< type > & other )  [inline]
```

Swaps the content of this list with another list.

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

- include/dsl/list.h

2.7 dsl::set< key, compare > Class Template Reference

```
#include <set.h>
```

Classes

- struct iterator

Public Member Functions

- set (const set &other)
- set & operator= (set other)
- void swap (set &other)
- iterator begin ()
- iterator end ()
- void insert (const key &key_value)
- iterator find (const key &key_value)
- iterator lower_bound (const key &value)
- iterator upper_bound (const key &value)
- void erase (iterator to_erase)
- size_t size () const
- bool empty () const
- void clear ()

2.7.1 Detailed Description

template<class key, class compare = std::less<key>>
class dsl::set< key, compare >

This class is an implementation of an ordered set using a treap data structure.

Template Parameters

key	The type of the value of an entry in the set.
compare	A binary predicate that defines a strict weak ordering, used to order the elements. The expression compare(a,b) shall return true if a is considered to go before b.

2.7.2 Constructor & Destructor Documentation

2.7.2.1 set()

```
template<class key , class compare = std::less<key>>
dsl::set< key, compare >::set (
             const set< key, compare > & other )  [inline]
```

Copy constructor, make a copy of the other set.

2.7.3 Member Function Documentation

2.7.3.1 begin()

```
template<class key , class compare = std::less<key>>
iterator dsl::set< key, compare >::begin ( )  [inline]
```

Returns an iterator to the first element in the set.

2.7.3.2 clear()

```
template<class key , class compare = std::less<key>>
void dsl::set< key, compare >::clear ( )  [inline]
```

Removes all the elements from the set.

2.7.3.3 empty()

```
template<class key , class compare = std::less<key>>
bool dsl::set< key, compare >::empty ( ) const  [inline]
```

Checks if the container is empty.

2.7.3.4 end()

```
template<class key , class compare = std::less<key>>
iterator dsl::set< key, compare >::end ( )  [inline]
```

Returns an iterator that represents the end of the set.

2.7.3.5 erase()

```
template<class key , class compare = std::less<key>>
void dsl::set< key, compare >::erase (
             iterator to_erase )  [inline]
```

Removes an element from the set by iterator.

2.7.3.6 find()

```
template<class key , class compare = std::less<key>>
iterator dsl::set< key, compare >::find (
             const key & key_value )  [inline]
```

Returns an iterator that points to the element with the given key. If no element with the given key is found in the set, return the end iterator.

2.7.3.7 insert()

```
template<class key , class compare = std::less<key>>
void dsl::set< key, compare >::insert (
             const key & key_value )  [inline]
```

Insert a new entry with the given key value.

2.7.3.8 lower_bound()

```
template<class key , class compare = std::less<key>>
iterator dsl::set< key, compare >::lower_bound (
             const key & value )  [inline]
```

Returns an iterator to the first element which is not considered to go before the given value.

2.7.3.9 operator=()

```
template<class key , class compare = std::less<key>>
set& dsl::set< key, compare >::operator= (
             set< key, compare > other )  [inline]
```

Assigns new contents to the set, replacing its current contents.

2.7.3.10 size()

```
template<class key , class compare = std::less<key>>
size_t dsl::set< key, compare >::size ( ) const  [inline]
```

Returns the number of elements in the set.

2.7.3.11 swap()

```
template<class key , class compare = std::less<key>>
void dsl::set< key, compare >::swap (
             set< key, compare > & other )  [inline]
```

Swaps the content of this set with another set.

2.7.3.12 upper_bound()

```
template<class key , class compare = std::less<key>>
iterator dsl::set< key, compare >::upper_bound (
             const key & value )  [inline]
```

Returns an iterator to the first element which is considered to go after the given value.

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

- include/dsl/set.h

Index