# 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 >**

Aceasta clasa este o implementare a unei tabele de dispersie ce foloseste o cautare liniara pentru a rezolva coliziuni.

Un element reprezinta o pereche de tip cheie-valoare.

Elementele sunt organizate in "bucket"-uri pentru a se stoca valorile in functie de "valoarea hash" a cheii.

Se foloseste clasa std::vector pentru obiectul de tip "bucket".

Un articol ce explica sumar conceptul de tabela hash se poate gasi la https://infoarena.↩ro/tabele-hash-scurta-prezentare.

**Template Parameters**

| | |
|---|---|
| *key* | Tipul cheii unui element in tabela de dispersie. |
| *value* | Tipul valorii unui element in tabela de dispersie. |
| *hash* | O clasa ce reprezinta o functie unara ce accepta un parametru de tip "key", folosita pentru a genera "valoarea hash" a unei chei. |
| *equal* | O clasa ce reprezinta o functie binara ce accepta doi parametrii de tip "key", folosita pentru a verifica daca doua chei sunt egale. |

### 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]
```

Returneaza un iterator ce reprezinta inceputul tabelei de dispersie, adica o referinta la primul element din tabela.

#### 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]
```

Sterge toate elementele din tabela de dispersie.

#### 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]
```

Verifica daca tabela de dispersie este goala.

#### 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]
```

Returneaza un iterator ce reprezinta sfarsitul tabelei de dispersie. Acest iterator nu trebuie accesat deoarece nu are o referinta la vreun element din tabela.

#### 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]
```

Sterge elementul dat de iteratorul luat ca parametru. Daca iteratorul nu este valid, comportamentul acestei metode nu este definit.

#### 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]
```

Returneaza un iterator la elementul ce are cheia data. Daca niciun element nu are cheia data, returneaza iteratorul ce marcheaza sfarsitul tabelei.

#### 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]
```

Insereaza un nou element in tabela de dispersie. Daca exista deja un element cu cheia data, valoarea sa nu este modificata.

#### 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]
```

Returneaza numarul de elemente din tabela de dispersie.

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 >**

Aceasta clasa este o implementare a unei cozi cu prioritati ce foloseste o structura de date de tip "heap".

Se foloseste clasa std::vector pentru a stoca elementele.

Un articol ce explica modul in care functioneaza structura de date de tip heap se poate gasi la https://infoarena.ro/heapuri.

**Template Parameters**

| | |
|---|---|
| *type* | Tipul valorii unui element in structura de tip "heap". |
| *compare* | O clasa ce reprezinta o functie binara ce defineste o ordonare de tip "mai mic strict", folosita pentru a ordona elementele.<br>Expresia compare(a,b) trebuie sa returneze true daca a ar trebui sa se afle inaintea lui 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]
```

Construieste coada cu prioritati prin inserarea tuturor elementelor in intervalul [first,last) si sortarea structurii "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]
```

Goleste coada cu prioritati.

#### 2.2.3.2 empty()

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

Verifica daca nu exista niciun element in coada cu prioritati.

#### 2.2.3.3 pop()

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

Sterge primul element din coada, adica elementul cu prioritate maxima.

#### 2.2.3.4 push()

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

Insereaza o noua valoare in coada cu prioritati.

#### 2.2.3.5 size()

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

Returneaza numarul de elemente din coada cu prioritati.

#### 2.2.3.6 top()

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

Returneaza valoarea primului element din coada, adica elementul cu prioritate maxima.

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**

Aceasta clasa este iteratorul setului. Parcurgerea setului returneaza elementele in ordinea definita de clasa comparator a setului.

### 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]
```

Obtine o referinta la un element din set.

#### 2.3.2.2 operator++() [1/2]

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

Muta iteratorul la succesorul elementului curent din set.

#### 2.3.2.3 operator++() [2/2]

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

Muta iteratorul la succesorul elementului curent din set.

#### 2.3.2.4 operator--() [1/2]

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

Muta iteratorul la predecesorul elementului curent din set.

#### 2.3.2.5 operator--() [2/2]

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

Muta iteratorul la predecesorul elementului curent din set.

#### 2.3.2.6 operator->()

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

Obtine o referinta la un element din set.

### 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]
```

Verifica daca doi iteratori nu sunt egali.

#### 2.3.3.2 operator==

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

Verifica daca doi iteratori sunt egali.

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**

Aceasta clasa este iteratorul tabelei de dispersie. Parcurgerea tabelei de dispersie returneaza elementele intr-o ordine aparent aleatorie.

### 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]
```

Obtine o referinta la un element din tabela de dispersie. Valoarea cheii este expusa doar ca nu trebuie modificata.

#### 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]
```

Gaseste urmatoarea valoare in tabela de dispersie, sarind peste "bucket"-urile goale.

#### 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]
```

Gaseste urmatoarea valoare in tabela de dispersie, sarind peste "bucket"-urile goale.

#### 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]
```

Obtine o referinta la un element din tabela de dispersie. Valoarea cheii este expusa doar ca nu trebuie modificata.

### 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]
```

Verifica daca doi iteratori nu sunt egali.

#### 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]
```

Verifica daca doi iteratori sunt egali.

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**

Aceasta clasa este iteratorul listei dublu inlantuite. Parcurgerea listei returneaza elementele in ordinea in care au fost inserate in lista.

### 2.5.2 Member Function Documentation

#### 2.5.2.1 operator*()

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

Obtine o referinta la un element din lista.

#### 2.5.2.2 operator++() [1/2]

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

Muta iteratorul la urmatorul element din lista.

#### 2.5.2.3 operator++() [2/2]

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

Muta iteratorul la urmatorul element din lista.

#### 2.5.2.4 operator--() [1/2]

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

Muta iteratorul la elementul precedent din lista.

#### 2.5.2.5 operator--() [2/2]

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

Muta iteratorul la elementul precedent din lista.

#### 2.5.2.6 operator->()

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

Obtine o referinta la un element din lista.

### 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]
```

Verifica daca doi iteratori nu sunt egali.

#### 2.5.3.2 operator==

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

Verifica daca doi iteratori sunt egali.

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 >**

Aceasta clasa este o implementare a unei liste dublu inlantuite.

**Template Parameters**

| | |
|---|---|
| *type* | Tipul valorii unui element din lista. |

### 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]
```

Constructor de copiere, creeaza o copie a celeilalte liste.

#### 2.6.2.2 list() [2/2]

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

Interschimba continutul acestei liste cu cel al listei de tip "rvalue".

#### 2.6.2.3 ~list()

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

Distruge obiectul lista.

### 2.6.3 Member Function Documentation

#### 2.6.3.1 back()

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

Returneaza o referinta la ultimul element al listei.

Comportamentul acestei functii apelate cand lista este goala este indefinit.

#### 2.6.3.2 begin()

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

Returneaza un iterator ce reprezinta inceputul listei.

#### 2.6.3.3 clear()

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

Goleste lista.

#### 2.6.3.4 empty()

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

Verifica daca nu exista niciun element in lista.

#### 2.6.3.5 end()

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

Returneaza un iterator ce reprezinta sfarsitul listei. Acest iterator nu trebuie accesat deoarece nu are o referinta la vreun element din lista.

#### 2.6.3.6 erase()

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

Sterge elementul de la pozitia specificata de iterator.

Returneaza un iterator la elementul ce era succesorul elementului sters.

#### 2.6.3.7 front()

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

Returneaza o referinta la primul element al listei.

Comportamentul acestei functii apelate cand lista este goala este indefinit.

#### 2.6.3.8 insert()

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

Insereaza un nou element cu valoarea data inainte de elementul de la pozitia specificata de iterator.

Returneaza un iterator la noul element inserat.

#### 2.6.3.9 operator=()

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

Atribuie un nou continut listei, prin intermediul altei liste. Copiaza continutul celeilalte liste in lista curenta.

#### 2.6.3.10 size()

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

Returneaza numarul de elemente din lista.

#### 2.6.3.11 swap()

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

Interschimba continutul acestei liste cu cel al listei date.

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 >**

Aceasta clasa este o implementare a unui set ordonat ce foloseste o structura de date de tip "treap".

Elementele din set sunt ordonate dupa valoarea cheii lor.

Un articol ce explica modul in care functioneaza structura de date de tip "treap" se poate gasi la https://www.infoarena.ro/treapuri.

**Template Parameters**

| | |
|---|---|
| *key* | Tipul valorii unui element din set. |
| *compare* | O clasa ce reprezinta o functie binara ce defineste o ordonare de tip "mai mic strict", folosita pentru a ordona elementele.<br>Expresia compare(a,b) trebuie sa returneze true daca a ar trebui sa se afle inaintea lui 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]
```

Constructor de copiere, creeaza o copie a celuilalt 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]
```

Returneaza un iterator ce reprezinta inceputul setului.

#### 2.7.3.2 clear()

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

Goleste setul.

#### 2.7.3.3 empty()

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

Verifica daca setul este gol.

#### 2.7.3.4 end()

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

Returneaza un iterator ce reprezinta sfarsitul setului. Acest iterator nu trebuie accesat deoarece nu are o referinta la vreun element din set.

#### 2.7.3.5 erase()

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

Sterge elementul de la pozitia specificata de 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]
```

Returneaza un iterator la elementul cu cheia data. Daca elementul nu este gasit, returneaza iteratorul ce marcheaza sfarsitul setului.

#### 2.7.3.7 insert()

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

Insereaza un nou element cu valoarea data in set.

#### 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]
```

Returneaza un iterator la primul element a carui valoare nu preceda valoarea data. Elementul poate avea o valoare egala, sau o valoare ce "merge dupa" valoarea data.

#### 2.7.3.9 operator=()

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

Atribuie un nou continut setului, prin intermediul altui set. Copiaza continutul celuilalt set in setul curent.

#### 2.7.3.10 size()

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

Returneaza numarul de elemente din 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]
```

Interschimba continutul acestui set cu cel al setului dat.

#### 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]
```

Returneaza un iterator la primul element a carui valoare "merge dupa" valoarea data.

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

- include/dsl/set.h

# Index