

cuckoo

Generated by Doxygen 1.6.3

Mon May 30 18:53:26 2011



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	cuckoo< Key, Value, Hash, Equal >::const_iterator Class Reference . . . . .	5
3.1.1	Detailed Description . . . . .	6
3.1.2	Constructor & Destructor Documentation . . . . .	6
3.1.2.1	const_iterator . . . . .	6
3.1.2.2	const_iterator . . . . .	6
3.1.2.3	const_iterator . . . . .	7
3.1.3	Member Function Documentation . . . . .	7
3.1.3.1	operator= . . . . .	7
3.1.3.2	operator++ . . . . .	7
3.1.3.3	operator++ . . . . .	7
3.1.3.4	operator* . . . . .	7
3.1.3.5	operator-> . . . . .	7
3.1.3.6	operator== . . . . .	7
3.1.3.7	operator!= . . . . .	7
3.1.4	Friends And Related Function Documentation . . . . .	7
3.1.4.1	cuckoo . . . . .	7
3.1.5	Member Data Documentation . . . . .	7
3.1.5.1	pos . . . . .	7
3.1.5.2	hash . . . . .	7
3.2	cuckoo< Key, Value, Hash, Equal > Class Template Reference . . . . .	8
3.2.1	Member Typedef Documentation . . . . .	10
3.2.1.1	Data . . . . .	10

3.2.2	Constructor & Destructor Documentation	10
3.2.2.1	cuckoo	10
3.2.2.2	~cuckoo	10
3.2.2.3	cuckoo	10
3.2.2.4	cuckoo	10
3.2.3	Member Function Documentation	11
3.2.3.1	get_exists	11
3.2.3.2	set_exists	11
3.2.3.3	unset_exists	11
3.2.3.4	init	11
3.2.3.5	copy	11
3.2.3.6	clear_all	12
3.2.3.7	data_from	12
3.2.3.8	is_here	12
3.2.3.9	hash	12
3.2.3.10	update_exists	12
3.2.3.11	update_data	13
3.2.3.12	rehash	13
3.2.3.13	add_new	13
3.2.3.14	remove	13
3.2.3.15	operator=	13
3.2.3.16	set_up	14
3.2.3.17	operator==	14
3.2.3.18	operator!=	14
3.2.3.19	swap	14
3.2.3.20	begin	14
3.2.3.21	begin	15
3.2.3.22	end	15
3.2.3.23	end	15
3.2.3.24	operator[]	15
3.2.3.25	erase	15
3.2.3.26	erase	16
3.2.3.27	erase	16
3.2.3.28	find	16
3.2.3.29	find	16
3.2.3.30	count	17

3.2.3.31	<a href="#">equal_range</a>	17
3.2.3.32	<a href="#">equal_range</a>	17
3.2.3.33	<a href="#">insert</a>	17
3.2.3.34	<a href="#">insert</a>	18
3.2.3.35	<a href="#">clear</a>	18
3.2.3.36	<a href="#">empty</a>	18
3.2.3.37	<a href="#">size</a>	18
3.2.3.38	<a href="#">length</a>	18
3.2.4	<a href="#">Friends And Related Function Documentation</a>	19
3.2.4.1	<a href="#">iterator</a>	19
3.2.4.2	<a href="#">const_iterator</a>	19
3.2.5	<a href="#">Member Data Documentation</a>	19
3.2.5.1	<a href="#">d_</a>	19
3.2.5.2	<a href="#">init_length_</a>	19
3.2.5.3	<a href="#">max_loop_</a>	19
3.2.5.4	<a href="#">step_</a>	19
3.2.5.5	<a href="#">hasher_</a>	19
3.2.5.6	<a href="#">key_equal_</a>	19
3.2.5.7	<a href="#">data_</a>	19
3.2.5.8	<a href="#">exists_</a>	20
3.2.5.9	<a href="#">len_</a>	20
3.2.5.10	<a href="#">len_part_</a>	20
3.2.5.11	<a href="#">size_</a>	20
3.2.5.12	<a href="#">is_rehashed_</a>	20
3.3	<a href="#">cuckoo&lt; Key, Value, Hash, Equal &gt;::iterator Class Reference</a>	21
3.3.1	<a href="#">Detailed Description</a>	21
3.3.2	<a href="#">Constructor &amp; Destructor Documentation</a>	21
3.3.2.1	<a href="#">iterator</a>	21
3.3.2.2	<a href="#">iterator</a>	21
3.3.2.3	<a href="#">iterator</a>	22
3.3.3	<a href="#">Member Function Documentation</a>	22
3.3.3.1	<a href="#">operator const_iterator</a>	22
3.3.3.2	<a href="#">operator=</a>	22
3.3.3.3	<a href="#">operator++</a>	22
3.3.3.4	<a href="#">operator++</a>	22
3.3.3.5	<a href="#">operator*</a>	22

---

3.3.3.6	operator->	22
3.3.3.7	operator==	22
3.3.3.8	operator!=	22
3.3.4	Friends And Related Function Documentation	22
3.3.4.1	cuckoo	22
3.3.5	Member Data Documentation	22
3.3.5.1	pos	22
3.3.5.2	hash	22
<b>4</b>	<b>File Documentation</b>	<b>23</b>
4.1	cuckoo.hpp File Reference	23
4.1.1	Detailed Description	23
4.1.2	LICENSE	23
4.1.3	DESCRIPTION	23
<b>5</b>	<b>Example Documentation</b>	<b>25</b>
5.1	cuckoo	25

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">cuckoo&lt; Key, Value, Hash, Equal &gt;::const_iterator</a> . . . . .	5
<a href="#">cuckoo&lt; Key, Value, Hash, Equal &gt;</a> . . . . .	8
<a href="#">cuckoo&lt; Key, Value, Hash, Equal &gt;::iterator</a> . . . . .	21





## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">cuckoo.hpp</a> . . . . .	23
--------------------------------------	----



## Chapter 3

# Class Documentation

### 3.1 cuckoo< Key, Value, Hash, Equal >::const\_iterator Class Reference

```
#include <cuckoo.hpp>
```

#### Public Member Functions

- [const\\_iterator](#) ()
- void [operator=](#) (const [const\\_iterator](#) &it)
- [const\\_iterator](#) (const [const\\_iterator](#) &it)
- [const\\_iterator](#) & [operator++](#) ()
- [const\\_iterator](#) [operator++](#) (int)
- const [Data](#) & [operator\\*](#) () const
- const [Data](#) \* [operator->](#) () const
- bool [operator==](#) (const [const\\_iterator](#) &it)
- bool [operator!=](#) (const [const\\_iterator](#) &it)

#### Private Member Functions

- [const\\_iterator](#) (const size\_t p, const [cuckoo](#) \*h)

#### Private Attributes

- size\_t [pos](#)
- const [cuckoo](#) \* [hash](#)

#### Friends

- class [cuckoo](#)

### 3.1.1 Detailed Description

```
template<class Key, class Value, class Hash, class Equal> class cuckoo< Key, Value, Hash, Equal
>::const_iterator
```

Used with const cuckoo objects.

### 3.1.2 Constructor & Destructor Documentation

**3.1.2.1** `template<class Key, class Value, class Hash, class Equal> cuckoo< Key, Value, Hash, Equal >::const_iterator::const_iterator (const size_t p, const cuckoo * h) [inline, private]`

**3.1.2.2** `template<class Key, class Value, class Hash, class Equal> cuckoo< Key, Value, Hash, Equal >::const_iterator::const_iterator () [inline]`

Default constructor.

**3.1.2.3** `template<class Key, class Value, class Hash, class Equal> cuckoo< Key, Value, Hash, Equal >::const_iterator::const_iterator (const const_iterator & it) [inline]`

### 3.1.3 Member Function Documentation

**3.1.3.1** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value, Hash, Equal >::const_iterator::operator= (const const_iterator & it) [inline]`

**3.1.3.2** `template<class Key, class Value, class Hash, class Equal> const_iterator& cuckoo< Key, Value, Hash, Equal >::const_iterator::operator++ () [inline]`

**3.1.3.3** `template<class Key, class Value, class Hash, class Equal> const_iterator cuckoo< Key, Value, Hash, Equal >::const_iterator::operator++ (int) [inline]`

**3.1.3.4** `template<class Key, class Value, class Hash, class Equal> const Data& cuckoo< Key, Value, Hash, Equal >::const_iterator::operator* () const [inline]`

**3.1.3.5** `template<class Key, class Value, class Hash, class Equal> const Data* cuckoo< Key, Value, Hash, Equal >::const_iterator::operator-> () const [inline]`

**3.1.3.6** `template<class Key, class Value, class Hash, class Equal> bool cuckoo< Key, Value, Hash, Equal >::const_iterator::operator== (const const_iterator & it) [inline]`

**3.1.3.7** `template<class Key, class Value, class Hash, class Equal> bool cuckoo< Key, Value, Hash, Equal >::const_iterator::operator!= (const const_iterator & it) [inline]`

### 3.1.4 Friends And Related Function Documentation

**3.1.4.1** `template<class Key, class Value, class Hash, class Equal> friend class cuckoo [friend]`

### 3.1.5 Member Data Documentation

**3.1.5.1** `template<class Key, class Value, class Hash, class Equal> size_t cuckoo< Key, Value, Hash, Equal >::const_iterator::pos [private]`

**3.1.5.2** `template<class Key, class Value, class Hash, class Equal> const cuckoo* cuckoo< Key, Value, Hash, Equal >::const_iterator::hash [private]`

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

- [cuckoo.hpp](#)

## 3.2 cuckoo< Key, Value, Hash, Equal > Class Template Reference

```
#include <cuckoo.hpp>
```

### Classes

- class [const\\_iterator](#)
- class [iterator](#)

### Public Member Functions

- [cuckoo](#) (size\_t d=D, size\_t init\_length=INIT\_LENGTH, size\_t max\_loop=MAX\_LOOP, double step=STEP, const Hash &hasher=Hash(), const Equal &equal=Equal())
- [~cuckoo](#) ()
- [cuckoo](#)< Key, Value, Hash, Equal > & [operator=](#) (const [cuckoo](#)< Key, Value, Hash, Equal > &Cuckoo)
- [cuckoo](#) (const [cuckoo](#)< Key, Value, Hash, Equal > &Cuckoo)
- template<class InputIterator >  
  [cuckoo](#) (InputIterator first, InputIterator last, const Hash &hasher=Hash(), const Equal &equal=Equal())
- void [set\\_up](#) (size\_t d=D, size\_t init\_length=INIT\_LENGTH, size\_t max\_loop=MAX\_LOOP, double step=STEP)
- bool [operator==](#) (const [cuckoo](#)< Key, Value, Hash, Equal > &Cuckoo)
- bool [operator!=](#) (const [cuckoo](#)< Key, Value, Hash, Equal > &Cuckoo)
- void [swap](#) ([cuckoo](#)< Key, Value, Hash, Equal > &Cuckoo)
- [iterator](#) [begin](#) ()
- [const\\_iterator](#) [begin](#) () const
- [iterator](#) [end](#) ()
- [const\\_iterator](#) [end](#) () const
- Value & [operator\[\]](#) (const Key &k)
- void [erase](#) ([iterator](#) it)
- void [erase](#) ([iterator](#) first, [iterator](#) last)
- size\_t [erase](#) (const Key &k)
- [iterator](#) [find](#) (const Key &k)
- [const\\_iterator](#) [find](#) (const Key &k) const
- size\_t [count](#) (const Key &k) const
- pair< [iterator](#), [iterator](#) > [equal\\_range](#) (const Key &k)
- pair< [const\\_iterator](#), [const\\_iterator](#) > [equal\\_range](#) (const Key &k) const
- pair< [iterator](#), bool > [insert](#) (const Data &k)
- template<class InputIterator >  
  void [insert](#) (InputIterator first, InputIterator last)
- void [clear](#) ()
- bool [empty](#) () const
- size\_t [size](#) () const
- size\_t [length](#) () const

### Private Types

- typedef pair< Key, Value > [Data](#)

## Private Member Functions

- bool [get\\_exists](#) (size\_t pos) const
- void [set\\_exists](#) (size\_t pos)
- void [unset\\_exists](#) (size\_t pos)
- void [init](#) ()
- void [copy](#) (const cuckoo< Key, Value, Hash, Equal > &Cuckoo)
- void [clear\\_all](#) ()
- [Data](#) & [data\\_from](#) (size\_t pos) const
- bool [is\\_here](#) (const Key &k, size\_t pos) const
- size\_t [hash](#) (const Key &k, size\_t hash\_num) const
- void [update\\_exists](#) (size\_t len\_temp\_)
- void [update\\_data](#) (size\_t len\_temp\_)
- void [rehash](#) ()
- size\_t [add\\_new](#) ([Data](#) p)
- [iterator](#) [remove](#) ([iterator](#) &it)

## Private Attributes

- size\_t [d\\_](#)
- size\_t [init\\_length\\_](#)
- size\_t [max\\_loop\\_](#)
- double [step\\_](#)
- Hash [hasher\\_](#)
- Equal [key\\_equal\\_](#)
- [Data](#) \*\* [data\\_](#)
- char \* [exists\\_](#)
- size\_t [len\\_](#)
- size\_t [len\\_part\\_](#)
- size\_t [size\\_](#)
- bool [is\\_rehashed\\_](#)

## Friends

- class [iterator](#)
- class [const\\_iterator](#)

```
template<class Key, class Value, class Hash, class Equal> class cuckoo< Key, Value, Hash, Equal >
```

### 3.2.1 Member Typedef Documentation

```
3.2.1.1 template<class Key, class Value, class Hash, class Equal> typedef pair<Key, Value>
cuckoo< Key, Value, Hash, Equal >::Data [private]
```

### 3.2.2 Constructor & Destructor Documentation

```
3.2.2.1 template<class Key, class Value, class Hash, class Equal> cuckoo< Key, Value, Hash,
Equal >::cuckoo (size_t d = D, size_t init_length = INIT_LENGTH, size_t max_loop =
MAX_LOOP, double step = STEP, const Hash & hasher = Hash (), const Equal & equal =
Equal ()) [inline, explicit]
```

Default constructor.

#### Parameters

- d* The number of hash functions (thus arrays also) that will be used in the program (can be  $\geq 2$ ).
- init\_length* The initial length of the whole structure. When you know the approximate number of records to be used, it is a good idea to take this value in 1.05-1.1 times more and small value of step.
- max\_loop* The maximum number of kick cycles during insertion before rehash.
- step* The ratio of increasing the size of hash during rehash. The less it is the less memory will be used but the more time is needed.
- hasher* The hash function object (template parameter by default).
- equal* The equal predictor object (template parameter by default).

```
3.2.2.2 template<class Key, class Value, class Hash, class Equal> cuckoo< Key, Value, Hash,
Equal >::~~cuckoo () [inline]
```

Destructor.

```
3.2.2.3 template<class Key, class Value, class Hash, class Equal> cuckoo< Key, Value, Hash,
Equal >::cuckoo (const cuckoo< Key, Value, Hash, Equal > & Cuckoo) [inline]
```

Copy constructor.

#### Parameters

- Cuckoo* The source of information

```
3.2.2.4 template<class Key, class Value, class Hash, class Equal> template<class InputIterator >
cuckoo< Key, Value, Hash, Equal >::cuckoo (InputIterator first, InputIterator last, const
Hash & hasher = Hash (), const Equal & equal = Equal ()) [inline]
```

Constructor from range [first, last).



**Parameters**

*first* The begin of range iterator.

*last* The end of range iterator.

**3.2.3 Member Function Documentation**

**3.2.3.1** `template<class Key, class Value, class Hash, class Equal> bool cuckoo< Key, Value, Hash, Equal >::get_exists (size_t pos) const [inline, private]`

Check whether there is a Data element at pos.

**Parameters**

*pos* Position in hash arrays.

**Returns**

true if some element presents on this position, false otherwise.

**3.2.3.2** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value, Hash, Equal >::set_exists (size_t pos) [inline, private]`

Set flag of existence of Data element at pos.

**Parameters**

*pos* Position in hash arrays.

**3.2.3.3** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value, Hash, Equal >::unset_exists (size_t pos) [inline, private]`

Unset flag of existence Data element at pos.

**Parameters**

*pos* Position in hash arrays.

**3.2.3.4** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value, Hash, Equal >::init () [inline, private]`

Initialize all the variables of cuckoo.

**3.2.3.5** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value, Hash, Equal >::copy (const cuckoo< Key, Value, Hash, Equal > & Cuckoo) [inline, private]`

Copy information from Cuckoo to clean object.

**Parameters**

*Object* for information to be copied from.

**3.2.3.6** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value, Hash, Equal >::clear_all () [inline, private]`

Clear all the data from cuckoo.

**3.2.3.7** `template<class Key, class Value, class Hash, class Equal> Data& cuckoo< Key, Value, Hash, Equal >::data_from (size_t pos) const [inline, private]`

Get Data element from pos.

#### Parameters

*pos* Position in hash arrays.

#### Returns

Reference to Data element.

**3.2.3.8** `template<class Key, class Value, class Hash, class Equal> bool cuckoo< Key, Value, Hash, Equal >::is_here (const Key & k, size_t pos) const [inline, private]`

Check whether element at pos has key k.

#### Parameters

*k* Key value.

*pos* Position in hash arrays.

#### Returns

true if element at pos is equal to k.

**3.2.3.9** `template<class Key, class Value, class Hash, class Equal> size_t cuckoo< Key, Value, Hash, Equal >::hash (const Key & k, size_t hash_num) const [inline, private]`

Return hash function result for key k.

#### Parameters

*k* Key value.

*hash\_num* The number of hash function from Hash family.

#### Returns

Hash value.

**3.2.3.10** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value, Hash, Equal >::update_exists (size_t len_temp_) [inline, private]`

Increase size of exists\_ up to len\_temp\_.

#### Parameters

*len\_temp\_* New size of exists\_.

**3.2.3.11** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value, Hash, Equal >::update_data (size_t len_temp_) [inline, private]`

Increase size of data\_ up to len\_temp\_.

#### Parameters

*len\_temp\_* New size of data\_.

**3.2.3.12** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value, Hash, Equal >::rehash () [inline, private]`

Rehash all the cuckoo (i.e. change size and replace Data elements).

**3.2.3.13** `template<class Key, class Value, class Hash, class Equal> size_t cuckoo< Key, Value, Hash, Equal >::add_new (Data p) [inline, private]`

Add new Data element.

#### Parameters

*p* New element.

#### Returns

0 to finish a recursion if it was needed.

**3.2.3.14** `template<class Key, class Value, class Hash, class Equal> iterator cuckoo< Key, Value, Hash, Equal >::remove (iterator &it) [inline, private]`

Remove element from cuckoo.

#### Parameters

*it* Iterator to element to be removed.

#### Returns

Iterator to next element after removed.

**3.2.3.15** `template<class Key, class Value, class Hash, class Equal> cuckoo<Key, Value, Hash, Equal>& cuckoo< Key, Value, Hash, Equal >::operator= (const cuckoo< Key, Value, Hash, Equal > &Cuckoo) [inline]`

Copy information from cuckoo of the same type.

#### Parameters

*Cuckoo* The source of data

**3.2.3.16** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value, Hash, Equal >::set_up (size_t d = D, size_t init_length = INIT_LENGTH, size_t max_loop = MAX_LOOP, double step = STEP) [inline]`

Update parameter of cuckoo, deleting all the data from it.

#### Warning

For test only!!!

**3.2.3.17** `template<class Key, class Value, class Hash, class Equal> bool cuckoo< Key, Value, Hash, Equal >::operator== (const cuckoo< Key, Value, Hash, Equal > & Cuckoo) [inline]`

Operator==

#### Parameters

*Cuckoo* Object to be compared with.

#### Returns

true if objects are equal (they are references to the same object).

**3.2.3.18** `template<class Key, class Value, class Hash, class Equal> bool cuckoo< Key, Value, Hash, Equal >::operator!= (const cuckoo< Key, Value, Hash, Equal > & Cuckoo) [inline]`

#### See also

operator==

**3.2.3.19** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value, Hash, Equal >::swap (cuckoo< Key, Value, Hash, Equal > & Cuckoo) [inline]`

Swap data with Cuckoo.

#### Parameters

*Cuckoo* Cuckoo of the same type

**3.2.3.20** `template<class Key, class Value, class Hash, class Equal> iterator cuckoo< Key, Value, Hash, Equal >::begin () [inline]`

Get iterator to begin of cuckoo.

#### Returns

Iterator to the first element of cuckoo.

**3.2.3.21** `template<class Key, class Value, class Hash, class Equal> const_iterator cuckoo< Key, Value, Hash, Equal >::begin () const [inline]`

Get [const\\_iterator](#) to begin of cuckoo (for const objects).

See also

[begin\(\)](#)

**3.2.3.22** `template<class Key, class Value, class Hash, class Equal> iterator cuckoo< Key, Value, Hash, Equal >::end () [inline]`

Get iterator to end of cuckoo.

**Returns**

Iterator to the position AFTER last element of cuckoo.

**3.2.3.23** `template<class Key, class Value, class Hash, class Equal> const_iterator cuckoo< Key, Value, Hash, Equal >::end () const [inline]`

Get [const\\_iterator](#) to begin of cuckoo (for const objects).

See also

[end\(\)](#)

**3.2.3.24** `template<class Key, class Value, class Hash, class Equal> Value& cuckoo< Key, Value, Hash, Equal >::operator[] (const Key & k) [inline]`

Get value by key or created pair key-value.

**Parameters**

*k* Key value.

**Returns**

Reference to Value, associated with k.

**3.2.3.25** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value, Hash, Equal >::erase (iterator it) [inline]`

Erase data at iterator.

**Parameters**

*it* Iterator to Data element to be removed.

**3.2.3.26** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value, Hash, Equal >::erase (iterator first, iterator last) [inline]`

Erase range of Data elements.

#### Parameters

*first* The begin of range iterator.

*last* The end of range iterator.

**3.2.3.27** `template<class Key, class Value, class Hash, class Equal> size_t cuckoo< Key, Value, Hash, Equal >::erase (const Key & k) [inline]`

Erase element by key.

#### Parameters

*k* Key value.

#### Returns

1 if element was erased and 0 if it didn't exist.

**3.2.3.28** `template<class Key, class Value, class Hash, class Equal> iterator cuckoo< Key, Value, Hash, Equal >::find (const Key & k) [inline]`

Find element by key.

#### Parameters

*k* Key value

#### Returns

Iterator to element or to end of cuckoo, if element doesn't exist.

**3.2.3.29** `template<class Key, class Value, class Hash, class Equal> const_iterator cuckoo< Key, Value, Hash, Equal >::find (const Key & k) const [inline]`

Find element by key (for const objects).

#### Parameters

*k* Key value

#### Returns

Const\_iterator to element or to end of cuckoo, if element doesn't exist.

**3.2.3.30** `template<class Key, class Value, class Hash, class Equal> size_t cuckoo< Key, Value, Hash, Equal >::count (const Key & k) const [inline]`

Count number of elements with this key.

#### Parameters

*k* Key value.

#### Returns

1 if element exists and 0 otherwise.

**3.2.3.31** `template<class Key, class Value, class Hash, class Equal> pair<iterator, iterator> cuckoo< Key, Value, Hash, Equal >::equal_range (const Key & k) [inline]`

Find range of elements with key.

#### Parameters

*k* Key value.

#### Returns

Pair that determines the range [first, last) or pair with both iterators pointing to the end of cuckoo.

**3.2.3.32** `template<class Key, class Value, class Hash, class Equal> pair<const_iterator, const_iterator> cuckoo< Key, Value, Hash, Equal >::equal_range (const Key & k) const [inline]`

Find range of elements with key (for const objects).

#### Parameters

*k* Key value

#### Returns

Pair that determines the range [first, last) or pair with both iterator pointing to the end of cuckoo.

**3.2.3.33** `template<class Key, class Value, class Hash, class Equal> pair<iterator, bool> cuckoo< Key, Value, Hash, Equal >::insert (const Data & k) [inline]`

Insert Data element to cuckoo.

#### Parameters

*k* The new Data element.

#### Returns

Pair with iterator to existing element and bool value, which is true if element was inserted or false if it existed before.

**3.2.3.34** `template<class Key, class Value, class Hash, class Equal> template<class InputIterator  
> void cuckoo< Key, Value, Hash, Equal >::insert (InputIterator first, InputIterator  
last) [inline]`

Insert range of Data elements.

#### Parameters

*first* The begin of range iterator.

*last* The end of range iterator.

**3.2.3.35** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value,  
Hash, Equal >::clear () [inline]`

Clear all data from cuckoo.

**3.2.3.36** `template<class Key, class Value, class Hash, class Equal> bool cuckoo< Key, Value,  
Hash, Equal >::empty () const [inline]`

Check whether cuckoo is empty.

#### Returns

true of cuckoo is empty and false otherwise.

**3.2.3.37** `template<class Key, class Value, class Hash, class Equal> size_t cuckoo< Key, Value,  
Hash, Equal >::size () const [inline]`

Show size of cuckoo.

#### Returns

Size of cuckoo.

**3.2.3.38** `template<class Key, class Value, class Hash, class Equal> size_t cuckoo< Key, Value,  
Hash, Equal >::length () const [inline]`

Show length of cuckoo (actual number of elements).

#### Returns

Length of cuckoo.



### 3.2.4 Friends And Related Function Documentation

**3.2.4.1** `template<class Key, class Value, class Hash, class Equal> friend class iterator` `[friend]`

**3.2.4.2** `template<class Key, class Value, class Hash, class Equal> friend class const_iterator`  
`[friend]`

### 3.2.5 Member Data Documentation

**3.2.5.1** `template<class Key, class Value, class Hash, class Equal> size_t cuckoo< Key, Value, Hash, Equal >::d_` `[private]`

The number of hash functions (thus arrays also) that will be used in the program (can be  $\geq 2$ ).

**3.2.5.2** `template<class Key, class Value, class Hash, class Equal> size_t cuckoo< Key, Value, Hash, Equal >::init_length_` `[private]`

The initial length of the whole structure. When you know the approximate number of records to be used, it is a good idea to take this value in 1.05-1.1 times more and small value of step.

**3.2.5.3** `template<class Key, class Value, class Hash, class Equal> size_t cuckoo< Key, Value, Hash, Equal >::max_loop_` `[private]`

The maximum number of kick cycles during insertion before rehash.

**3.2.5.4** `template<class Key, class Value, class Hash, class Equal> double cuckoo< Key, Value, Hash, Equal >::step_` `[private]`

The ratio of increasing the size of hash during rehash. The less it is the less memory will be used but the more time is needed.

**3.2.5.5** `template<class Key, class Value, class Hash, class Equal> Hash cuckoo< Key, Value, Hash, Equal >::hasher_` `[private]`

The hash function object (template parameter by default).

**3.2.5.6** `template<class Key, class Value, class Hash, class Equal> Equal cuckoo< Key, Value, Hash, Equal >::key_equal_` `[private]`

The equal predicator object (template parameter by default).

**3.2.5.7** `template<class Key, class Value, class Hash, class Equal> Data** cuckoo< Key, Value, Hash, Equal >::data_` `[private]`

The array of vectors, each of which is hash array.

**3.2.5.8** `template<class Key, class Value, class Hash, class Equal> char* cuckoo< Key, Value, Hash, Equal >::exists_ [private]`

The array of flags indicating existence of the element in hash.

**3.2.5.9** `template<class Key, class Value, class Hash, class Equal> size_t cuckoo< Key, Value, Hash, Equal >::len_ [private]`

The total length of all the hash arrays.

**3.2.5.10** `template<class Key, class Value, class Hash, class Equal> size_t cuckoo< Key, Value, Hash, Equal >::len_part_ [private]`

The length of every hash array.

**3.2.5.11** `template<class Key, class Value, class Hash, class Equal> size_t cuckoo< Key, Value, Hash, Equal >::size_ [private]`

The actual number of elements in cuckoo hash.

**3.2.5.12** `template<class Key, class Value, class Hash, class Equal> bool cuckoo< Key, Value, Hash, Equal >::is_rehashed_ [private]`

The flag that announces that rehash was made recently.

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

- [cuckoo.hpp](#)

### 3.3 cuckoo< Key, Value, Hash, Equal >::iterator Class Reference

```
#include <cuckoo.hpp>
```

#### Public Member Functions

- [iterator](#) ()
- [operator const\\_iterator](#) ()
- void [operator=](#) (const [iterator](#) &it)
- [iterator](#) (const [iterator](#) &it)
- [iterator](#) & [operator++](#) ()
- [iterator](#) [operator++](#) (int)
- [Data](#) & [operator\\*](#) ()
- [Data](#) \* [operator->](#) ()
- bool [operator==](#) (const [iterator](#) &it)
- bool [operator!=](#) (const [iterator](#) &it)

#### Private Member Functions

- [iterator](#) (size\_t p, [cuckoo](#) \*h)

#### Private Attributes

- size\_t [pos](#)
- [cuckoo](#) \* [hash](#)

#### Friends

- class [cuckoo](#)

#### 3.3.1 Detailed Description

**template<class Key, class Value, class Hash, class Equal> class cuckoo< Key, Value, Hash, Equal >::iterator**

Used with non-const cuckoo objects.

#### 3.3.2 Constructor & Destructor Documentation

**3.3.2.1** **template<class Key, class Value, class Hash, class Equal> cuckoo< Key, Value, Hash, Equal >::iterator::iterator (size\_t p, cuckoo \* h) [inline, private]**

**3.3.2.2** **template<class Key, class Value, class Hash, class Equal> cuckoo< Key, Value, Hash, Equal >::iterator::iterator () [inline]**

Default constructor.

**3.3.2.3** `template<class Key, class Value, class Hash, class Equal> cuckoo< Key, Value, Hash, Equal >::iterator::iterator (const iterator & it) [inline]`

### 3.3.3 Member Function Documentation

**3.3.3.1** `template<class Key, class Value, class Hash, class Equal> cuckoo< Key, Value, Hash, Equal >::iterator::operator const_iterator () [inline]`

Conversion to [const\\_iterator](#).

**3.3.3.2** `template<class Key, class Value, class Hash, class Equal> void cuckoo< Key, Value, Hash, Equal >::iterator::operator= (const iterator & it) [inline]`

**3.3.3.3** `template<class Key, class Value, class Hash, class Equal> iterator& cuckoo< Key, Value, Hash, Equal >::iterator::operator++ () [inline]`

**3.3.3.4** `template<class Key, class Value, class Hash, class Equal> iterator cuckoo< Key, Value, Hash, Equal >::iterator::operator++ (int) [inline]`

**3.3.3.5** `template<class Key, class Value, class Hash, class Equal> Data& cuckoo< Key, Value, Hash, Equal >::iterator::operator* () [inline]`

**3.3.3.6** `template<class Key, class Value, class Hash, class Equal> Data* cuckoo< Key, Value, Hash, Equal >::iterator::operator-> () [inline]`

**3.3.3.7** `template<class Key, class Value, class Hash, class Equal> bool cuckoo< Key, Value, Hash, Equal >::iterator::operator== (const iterator & it) [inline]`

**3.3.3.8** `template<class Key, class Value, class Hash, class Equal> bool cuckoo< Key, Value, Hash, Equal >::iterator::operator!= (const iterator & it) [inline]`

### 3.3.4 Friends And Related Function Documentation

**3.3.4.1** `template<class Key, class Value, class Hash, class Equal> friend class cuckoo [friend]`

### 3.3.5 Member Data Documentation

**3.3.5.1** `template<class Key, class Value, class Hash, class Equal> size_t cuckoo< Key, Value, Hash, Equal >::iterator::pos [private]`

**3.3.5.2** `template<class Key, class Value, class Hash, class Equal> cuckoo* cuckoo< Key, Value, Hash, Equal >::iterator::hash [private]`

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

- [cuckoo.hpp](#)

# Chapter 4

## File Documentation

### 4.1 cuckoo.hpp File Reference

```
#include <cstring>
```

#### Classes

- class cuckoo< Key, Value, Hash, Equal >
- class cuckoo< Key, Value, Hash, Equal >::const\_iterator
- class cuckoo< Key, Value, Hash, Equal >::iterator

#### 4.1.1 Detailed Description

##### Author

Mariya Fomkina

##### Version

1.0

#### 4.1.2 LICENSE

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

#### 4.1.3 DESCRIPTION

Cuckoo hashing implementation with the interface close to standard std::map and std::unordered\_map interface.

See [http://en.wikipedia.org/wiki/Cuckoo\\_hashing](http://en.wikipedia.org/wiki/Cuckoo_hashing)



## Chapter 5

# Example Documentation

### 5.1 cuckoo

#### Parameters

*Key* key type in hash.

*Value* value type in hash.

*Hash* function that gets data of type Key and some number (which is the number of appropriate hash function) and returns size\_t (e.g. Hash(int, size\_t)).

*Equal* predictor that compares two Key values. <int, int, Hasher, std::key\_equal\_to<int> >  
Cuckoo;

# Index

- [~cuckoo](#)
    - [cuckoo](#), [10](#)
- [add\\_new](#)
  - [cuckoo](#), [13](#)
- [begin](#)
  - [cuckoo](#), [14](#)
- [clear](#)
  - [cuckoo](#), [18](#)
- [clear\\_all](#)
  - [cuckoo](#), [11](#)
- [const\\_iterator](#)
  - [cuckoo](#), [19](#)
  - [cuckoo::const\\_iterator](#), [6](#)
- [copy](#)
  - [cuckoo](#), [11](#)
- [count](#)
  - [cuckoo](#), [16](#)
- [cuckoo](#), [8](#)
  - [~cuckoo](#), [10](#)
  - [add\\_new](#), [13](#)
  - [begin](#), [14](#)
  - [clear](#), [18](#)
  - [clear\\_all](#), [11](#)
  - [const\\_iterator](#), [19](#)
  - [copy](#), [11](#)
  - [count](#), [16](#)
  - [cuckoo](#), [10](#)
  - [cuckoo::const\\_iterator](#), [7](#)
  - [cuckoo::iterator](#), [22](#)
  - [d\\_](#), [19](#)
  - [Data](#), [10](#)
  - [data\\_](#), [19](#)
  - [data\\_from](#), [12](#)
  - [empty](#), [18](#)
  - [end](#), [15](#)
  - [equal\\_range](#), [17](#)
  - [erase](#), [15](#), [16](#)
  - [exists\\_](#), [19](#)
  - [find](#), [16](#)
  - [get\\_exists](#), [11](#)
  - [hash](#), [12](#)
  - [hasher\\_](#), [19](#)
  - [init](#), [11](#)
  - [init\\_length\\_](#), [19](#)
  - [insert](#), [17](#)
  - [is\\_here](#), [12](#)
  - [is\\_rehashed\\_](#), [20](#)
  - [iterator](#), [19](#)
  - [key\\_equal\\_](#), [19](#)
  - [len\\_](#), [20](#)
  - [len\\_part\\_](#), [20](#)
  - [length](#), [18](#)
  - [max\\_loop\\_](#), [19](#)
  - [operator=](#), [13](#)
  - [operator==](#), [14](#)
  - [rehash](#), [13](#)
  - [remove](#), [13](#)
  - [set\\_exists](#), [11](#)
  - [set\\_up](#), [13](#)
  - [size](#), [18](#)
  - [size\\_](#), [20](#)
  - [step\\_](#), [19](#)
  - [swap](#), [14](#)
  - [unset\\_exists](#), [11](#)
  - [update\\_data](#), [12](#)
  - [update\\_exists](#), [12](#)
- [cuckoo.hpp](#), [23](#)
- [cuckoo::const\\_iterator](#), [5](#)
  - [const\\_iterator](#), [6](#)
  - [cuckoo](#), [7](#)
  - [hash](#), [7](#)
  - [operator\\*](#), [7](#)
  - [operator++](#), [7](#)
  - [operator->](#), [7](#)
  - [operator=](#), [7](#)
  - [operator==](#), [7](#)
  - [pos](#), [7](#)
- [cuckoo::iterator](#), [21](#)
  - [cuckoo](#), [22](#)
  - [hash](#), [22](#)
  - [iterator](#), [21](#)
  - [operator const\\_iterator](#), [22](#)
  - [operator\\*](#), [22](#)
  - [operator++](#), [22](#)
  - [operator->](#), [22](#)
  - [operator=](#), [22](#)
  - [operator==](#), [22](#)



pos, 22

d\_  
    cuckoo, 19

Data  
    cuckoo, 10

data\_  
    cuckoo, 19

data\_from  
    cuckoo, 12

empty  
    cuckoo, 18

end  
    cuckoo, 15

equal\_range  
    cuckoo, 17

erase  
    cuckoo, 15, 16

exists\_  
    cuckoo, 19

find  
    cuckoo, 16

get\_exists  
    cuckoo, 11

hash  
    cuckoo, 12  
    cuckoo::const\_iterator, 7  
    cuckoo::iterator, 22

hasher\_  
    cuckoo, 19

init  
    cuckoo, 11

init\_length\_  
    cuckoo, 19

insert  
    cuckoo, 17

is\_here  
    cuckoo, 12

is\_rehashed\_  
    cuckoo, 20

iterator  
    cuckoo, 19  
    cuckoo::iterator, 21

key\_equal\_  
    cuckoo, 19

len\_  
    cuckoo, 20

len\_part\_  
    cuckoo, 20

length  
    cuckoo, 18

max\_loop\_  
    cuckoo, 19

operator const\_iterator  
    cuckoo::iterator, 22

operator\*  
    cuckoo::const\_iterator, 7  
    cuckoo::iterator, 22

operator++  
    cuckoo::const\_iterator, 7  
    cuckoo::iterator, 22

operator->  
    cuckoo::const\_iterator, 7  
    cuckoo::iterator, 22

operator=  
    cuckoo, 13  
    cuckoo::const\_iterator, 7  
    cuckoo::iterator, 22

operator==  
    cuckoo, 14  
    cuckoo::const\_iterator, 7  
    cuckoo::iterator, 22

pos  
    cuckoo::const\_iterator, 7  
    cuckoo::iterator, 22

rehash  
    cuckoo, 13

remove  
    cuckoo, 13

set\_exists  
    cuckoo, 11

set\_up  
    cuckoo, 13

size  
    cuckoo, 18

size\_  
    cuckoo, 20

step\_  
    cuckoo, 19

swap  
    cuckoo, 14

unset\_exists  
    cuckoo, 11

update\_data  
    cuckoo, 12

update\_exists  
    cuckoo, 12