Ring< Key, Info > Class Template Reference

Simple implementation of doubly-linked ring. Focuses mostly around insertion of the elements as well as iterators. More...

Classes

class	**iterator**

Public Member Functions

	Ring (const **Ring**< Key, Info > &copy)
Ring< Key, Info > &	**operator=** (const **Ring**< Key, Info > &copy)
bool	**operator==** (const **iterator** &it) const
bool	**operator!=** (const **iterator** &it) const
iterator	**begin** () const
bool	**empty** () const
void	**clear** ()
void	**push** (const Key &k, const Info &i)
void	**push** (const **iterator** &it)
bool	**push_after** (const Key &k, const Info &i, const Key &key, int occurrence=1)
bool	**push_after** (const **iterator** &it, const Key &key, int occurrence=1)
bool	**push_after** (const **iterator** &it, const **iterator** &ringElement)
bool	**push_before** (const Key &k, const Info &i, const Key &key, int occurrence=1)
bool	**push_before** (const **iterator** &it, const Key &key, int occurrence=1)
bool	**push_before** (const **iterator** &it, const **iterator** &ringElement)
bool	**remove** (const **iterator** &it)
bool	**find** (const Key &key, int occurrence, **iterator** &it, bool forwardDirection=true)
void	**show** (bool forwardDirection=true, std::ostream &out=std::cout)

Detailed Description

template<typename Key, typename Info>
class Ring< Key, Info >

Simple implementation of doubly-linked ring. Focuses mostly around insertion of the elements as well as iterators.

Member Function Documentation

◆ begin()

template<typename Key , typename Info >
Ring< Key, Info >::**iterator** **Ring**< Key, Info >::begin

Outputs the iterator to the first element of a **Ring**.

Returns
: Iterator(head);

◆ clear()

template<typename Key , typename Info >
void **Ring**< Key, Info >::clear

Clears the ring.

◆ empty()

template<typename Key , typename Info >

bool **Ring**< Key, Info >::empty

Outputs the information about the ring being empty.

Returns

: True - **Ring** is empty. False - otherwise.

◆ find()

template<typename Key , typename Info >

```
bool Ring< Key, Info >::find ( const Key &  key,
                               int          occurrence,
                               iterator &   it,
                               bool         forwardDirection = true
                             )
```

Find a given occurrence of a key in a ring.

Parameters

[in]	**key**	: A wanted key.
[in]	**occurrence**	: A occurrence of a wanted key.
[in]	**it**	: A iterator to which the node of the key is going to be assigned.
[in]	**forwardDirection**	: If true the direction is forward == node = node -> next, if false node = node -> prev

Returns

: True - the element was found. False - otherwise

◆ operator!=()

template<typename Key , typename Info >

bool **Ring**< Key, Info >::operator!= (const **iterator** & it) const inline

Inequality operator.

Returns

: True - iterator stores node of the **Ring**, False - otherwise.

◆ operator=()

template<typename Key , typename Info >

Ring< Key, Info > & **Ring**< Key, Info >::operator= (const **Ring**< Key, Info > & copy)

Assign operator.

◆ operator==()

template<typename Key , typename Info >

bool **Ring**< Key, Info >::operator== (const **iterator** & it) const inline

Equality operator.

Returns

: True - iterator stores node of the **Ring**, False - otherwise.

◆ push() [1/2]

template<typename Key , typename Info >

void **Ring**< Key, Info >::push (const **iterator** & it)

Method pushes the key and info of the iterator before any.

Parameters

[in] **it** : Iterator to the **Ring**.

◆ push() [2/2]

template<typename Key , typename Info >

void **Ring**< Key, Info >::push (const Key & k,
const Info & i
)

Method pushes the element before any. If any exists.

Parameters

[in] **k** : Key

[in] **i** : Info

◆ push_after() [1/3]

template<typename Key , typename Info >

bool **Ring**< Key, Info >::push_after (const **iterator** & it,
const **iterator** & ringElement
)

Itserts the given key and info of the iterator after the iterator of a 'this' **Ring**.

Parameters

[in] **it** : iterator to random **Ring**.

[in] **ringElement** : An element of the **Ring**.

Returns

: True - the element was successfully inserted, False - there was an error. it is null or ringElement is not a member of this.

◆ push_after() [2/3]

template<typename Key , typename Info >

bool **Ring**< Key, Info >::push_after (const **iterator** & it,
const Key & key,
int occurrence = 1
)

Itserts the given key and info of the iterator after the key.

Parameters

[in] **it** : iterator to random **Ring**.

[in] **key** : Node is going to be inserted after this key.

[in] **occurrence** : Key is not unique so we do need to take a occurrence in a count.

Returns

: True - element was successfully inserted, False - otherwise.

◆ push_after() [3/3]

```
template<typename Key , typename Info >
bool Ring< Key, Info >::push_after ( const Key &  k,
                                     const Info &  i,
                                     const Key &  key,
                                     int  occurrence = 1
                                   )
```

Itserts the given key and info after the key.

Parameters

[in]	**k**	: Key which is going to be inserted.
[in]	**i**	: Info which is going to be inserted.
[in]	**key**	: Node is going to be inserted after this key.
[in]	**occurrence**	: Key is not unique so we do need to take a occurrence in a count.

Returns

: True - element was successfully inserted, False - otherwise.

◆ push_before() [1/3]

```
template<typename Key , typename Info >
bool Ring< Key, Info >::push_before ( const iterator &  it,
                                      const iterator &  ringElement
                                    )
```

Itserts the given key and info of the iterator before the iterator of a 'this' Ring.

Parameters

[in]	**it**	: iterator to random Ring.
[in]	**ringElement**	: An element of the Ring.

Returns

: True - the element was successfully inserted, False - there was an error. it is null or ringElement is not a member of this.

◆ push_before() [2/3]

```
template<typename Key , typename Info >
bool Ring< Key, Info >::push_before ( const iterator &  it,
                                      const Key &  key,
                                      int  occurrence = 1
                                    )
```

Itserts the given key and info of the iterator before the key.

Parameters

[in]	**it**	: iterator to random Ring.
[in]	**key**	: Node is going to be inserted before this key.
[in]	**occurrence**	: Key is not unique so we do need to take a occurrence in a count.

Returns

: True - element was successfully inserted, False - otherwise.

◆ push_before() [3/3]

template<typename Key , typename Info >

bool **Ring**< Key, Info >::push_before (const Key & k,
const Info & i,
const Key & key,
int occurrence = 1
)

Itserts the given key and info before the key.

Parameters

[in] **k** : Key which is going to be inserted.

[in] **i** : Info which is going to be inserted.

[in] **key** : Node is going to be inserted before this key.

[in] **occurrence** : Key is not unique so we do need to take a occurrence in a count.

Returns

: True - element was successfully inserted, False - otherwise.

◆ remove()

template<typename Key , typename Info >

bool **Ring**< Key, Info >::remove (const **iterator** & it)

Removes a node to which iterator is pointing. If: iterator is pointing to a node from the current **Ring** == **Ring** is nonempty iterator is not null

Returns

: True - element was successfully deleted, False = otherwise.

◆ show()

template<typename Key , typename Info >

void **Ring**< Key, Info >::show (bool forwardDirection = `true`,
std::ostream & out = `std::cout`
)

Simple output method.

Parameters

[in] **forwardDirection** : True - direction set to forward. False - otherwise

[in] **out** : std::ostream type.