

Circular Linked List

Generated by Doxygen 1.9.1

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 CircularLinkedList< T > Class Template Reference	5
3.1.1 Detailed Description	6
3.1.2 Constructor & Destructor Documentation	7
3.1.2.1 CircularLinkedList() [1/2]	7
3.1.2.2 ~CircularLinkedList()	7
3.1.2.3 CircularLinkedList() [2/2]	7
3.1.3 Member Function Documentation	7
3.1.3.1 begin() [1/2]	7
3.1.3.2 begin() [2/2]	8
3.1.3.3 cbegin()	8
3.1.3.4 cend()	8
3.1.3.5 clear()	8
3.1.3.6 empty()	9
3.1.3.7 end() [1/2]	9
3.1.3.8 end() [2/2]	9
3.1.3.9 erase_after()	9
3.1.3.10 front() [1/2]	10
3.1.3.11 front() [2/2]	10
3.1.3.12 insert_after()	11
3.1.3.13 operator!=()	11
3.1.3.14 operator=()	11
3.1.3.15 operator==()	12
3.1.3.16 pop_front()	12
3.1.3.17 push_front()	12
3.1.3.18 size()	13
3.1.4 Friends And Related Function Documentation	13
3.1.4.1 const_iterator	13
3.1.4.2 iterator	13
3.2 CircularLinkedList< T >::const_iterator Class Reference	13
3.2.1 Detailed Description	14
3.2.2 Member Typedef Documentation	14
3.2.2.1 difference_type	15
3.2.2.2 iterator_category	15
3.2.2.3 pointer	15
3.2.2.4 reference	15
3.2.2.5 value_type	15

3.2.3 Constructor & Destructor Documentation	15
3.2.3.1 const_iterator() [1/2]	15
3.2.3.2 ~const_iterator()	16
3.2.3.3 const_iterator() [2/2]	16
3.2.4 Member Function Documentation	16
3.2.4.1 operator!=(())	16
3.2.4.2 operator*()	16
3.2.4.3 operator++() [1/2]	17
3.2.4.4 operator++() [2/2]	17
3.2.4.5 operator->()	18
3.2.4.6 operator=()	18
3.2.4.7 operator==(())	18
3.2.5 Friends And Related Function Documentation	18
3.2.5.1 CircularLinkedList	19
3.3 CircularLinkedList< T >::iterator Class Reference	19
3.3.1 Detailed Description	20
3.3.2 Member Typedef Documentation	20
3.3.2.1 difference_type	20
3.3.2.2 iterator_category	20
3.3.2.3 pointer	20
3.3.2.4 reference	20
3.3.2.5 value_type	20
3.3.3 Constructor & Destructor Documentation	21
3.3.3.1 iterator() [1/2]	21
3.3.3.2 ~iterator()	21
3.3.3.3 iterator() [2/2]	21
3.3.4 Member Function Documentation	21
3.3.4.1 operator!=(())	21
3.3.4.2 operator*()	22
3.3.4.3 operator++() [1/2]	22
3.3.4.4 operator++() [2/2]	22
3.3.4.5 operator->()	23
3.3.4.6 operator=()	23
3.3.4.7 operator==(())	23
3.3.5 Friends And Related Function Documentation	24
3.3.5.1 CircularLinkedList	24
4 File Documentation	25
4.1 circular_list.h File Reference	25
4.2 test.cpp File Reference	26
4.2.1 Function Documentation	26
4.2.1.1 main()	27

4.2.1.2 TEST() [1/10]	27
4.2.1.3 TEST() [2/10]	27
4.2.1.4 TEST() [3/10]	27
4.2.1.5 TEST() [4/10]	27
4.2.1.6 TEST() [5/10]	27
4.2.1.7 TEST() [6/10]	28
4.2.1.8 TEST() [7/10]	28
4.2.1.9 TEST() [8/10]	28
4.2.1.10 TEST() [9/10]	28
4.2.1.11 TEST() [10/10]	28

Index**29**

Chapter 1

Class Index

1.1 Class List

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

CircularLinkedList< T >	
A circular singly linked list container	5
CircularLinkedList< T >::const_iterator	
Constant forward iterator for CircularLinkedList	13
CircularLinkedList< T >::iterator	
Forward iterator for CircularLinkedList	19

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

circular_list.h	25
test.cpp	26

Chapter 3

Class Documentation

3.1 CircularLinkedList< T > Class Template Reference

A circular singly linked list container.

```
#include <circular_list.h>
```

Classes

- class [const_iterator](#)
Constant forward iterator for [CircularLinkedList](#).
- class [iterator](#)
Forward iterator for [CircularLinkedList](#).

Public Member Functions

- [CircularLinkedList](#) ()
Default constructor. Creates an empty [CircularLinkedList](#).
- [~CircularLinkedList](#) ()
Destructor. Clears the list and releases memory.
- [iterator begin](#) ()
Returns an iterator to the first element.
- [iterator end](#) ()
Returns an iterator to one past the last element.
- [const_iterator begin](#) () const
Returns a [const_iterator](#) to the first element.
- [const_iterator end](#) () const
Returns a [const_iterator](#) to one past the last element.
- [const_iterator cbegin](#) () const
Returns a [const_iterator](#) to the first element.
- [const_iterator cend](#) () const
Returns a [const_iterator](#) to one past the last element.
- void [push_front](#) (const T &value)
Inserts a new element at the front of the list.

- void `pop_front()`
Removes the first element of the list.
- T & `front()`
Accesses the first element.
- const T & `front()` const
Accesses the first element (const version).
- bool `empty()` const
Checks whether the list is empty.
- size_t `size()` const
Returns the number of elements in the list.
- iterator `insert_after` (const iterator pos, const T &value)
Inserts a new element after the given position.
- iterator `erase_after` (const iterator pos)
Erases the element following the given position.
- void `clear()`
Clears the list, deleting all elements.
- CircularLinkedList (const CircularLinkedList &other)
Copy constructor. Creates a deep copy of another CircularLinkedList.
- CircularLinkedList & `operator=` (const CircularLinkedList &other)
Copy assignment operator. Replaces the contents with a copy of another list.
- bool `operator==` (const CircularLinkedList &other) const
Equality comparison operator. Checks if two circular lists contain the same elements in the same order, accounting for circular rotation.
- bool `operator!=` (const CircularLinkedList &other) const
Inequality comparison operator.

Friends

- class `iterator`
Forward declaration of iterator classes as friends.
- class `const_iterator`

3.1.1 Detailed Description

```
template<typename T>
class CircularLinkedList< T >
```

A circular singly linked list container.

This class implements a circular singly linked list similar in interface to `std::forward_list`. The list elements are linked in a circle, where the last element points back to the head.

Template Parameters

<i>T</i>	Type of elements stored in the list.
----------	--------------------------------------

3.1.2 Constructor & Destructor Documentation

3.1.2.1 CircularLinkedList() [1/2]

```
template<typename T >
CircularLinkedList< T >::CircularLinkedList ( ) [inline]
```

Default constructor. Creates an empty [CircularLinkedList](#).

3.1.2.2 ~CircularLinkedList()

```
template<typename T >
CircularLinkedList< T >::~~CircularLinkedList ( ) [inline]
```

Destructor. Clears the list and releases memory.

3.1.2.3 CircularLinkedList() [2/2]

```
template<typename T >
CircularLinkedList< T >::CircularLinkedList (
    const CircularLinkedList< T > & other ) [inline]
```

Copy constructor. Creates a deep copy of another [CircularLinkedList](#).

Parameters

<i>other</i>	The list to copy from.
--------------	------------------------

3.1.3 Member Function Documentation

3.1.3.1 begin() [1/2]

```
template<typename T >
iterator CircularLinkedList< T >::begin ( ) [inline]
```

Returns an iterator to the first element.

Returns

iterator pointing to the head element.

3.1.3.2 begin() [2/2]

```
template<typename T >
const_iterator CircularLinkedList< T >::begin ( ) const [inline]
```

Returns a [const_iterator](#) to the first element.

Returns

[const_iterator](#) pointing to the head element.

3.1.3.3 cbegin()

```
template<typename T >
const_iterator CircularLinkedList< T >::cbegin ( ) const [inline]
```

Returns a [const_iterator](#) to the first element.

Returns

[const_iterator](#) pointing to the head element.

3.1.3.4 cend()

```
template<typename T >
const_iterator CircularLinkedList< T >::cend ( ) const [inline]
```

Returns a [const_iterator](#) to one past the last element.

Returns

[const_iterator](#) representing the end of the list.

3.1.3.5 clear()

```
template<typename T >
void CircularLinkedList< T >::clear ( ) [inline]
```

Clears the list, deleting all elements.

3.1.3.6 empty()

```
template<typename T >
bool CircularLinkedList< T >::empty ( ) const [inline]
```

Checks whether the list is empty.

Returns

True if the list contains no elements.

3.1.3.7 end() [1/2]

```
template<typename T >
iterator CircularLinkedList< T >::end ( ) [inline]
```

Returns an iterator to one past the last element.

Returns

iterator representing the end of the list.

3.1.3.8 end() [2/2]

```
template<typename T >
const_iterator CircularLinkedList< T >::end ( ) const [inline]
```

Returns a [const_iterator](#) to one past the last element.

Returns

[const_iterator](#) representing the end of the list.

3.1.3.9 erase_after()

```
template<typename T >
iterator CircularLinkedList< T >::erase_after (
    const iterator pos ) [inline]
```

Erases the element following the given position.

Parameters

<i>pos</i>	Iterator pointing to the element before the one to erase.
------------	---

Returns

Iterator pointing to the element following the erased element.

Exceptions

<i>std::invalid_argument</i>	if pos is invalid or nothing to erase.
------------------------------	--

3.1.3.10 front() [1/2]

```
template<typename T >
T& CircularLinkedList< T >::front ( ) [inline]
```

Accesses the first element.

Returns

Reference to the first element's value.

Exceptions

<i>std::out_of_range</i>	if the list is empty.
--------------------------	-----------------------

3.1.3.11 front() [2/2]

```
template<typename T >
const T& CircularLinkedList< T >::front ( ) const [inline]
```

Accesses the first element (const version).

Returns

Const reference to the first element's value.

Exceptions

<i>std::out_of_range</i>	if the list is empty.
--------------------------	-----------------------

3.1.3.12 insert_after()

```
template<typename T >
iterator CircularLinkedList< T >::insert_after (
    const iterator pos,
    const T & value ) [inline]
```

Inserts a new element after the given position.

Parameters

<i>pos</i>	Iterator pointing to the element after which to insert.
<i>value</i>	Value to insert.

Returns

Iterator pointing to the newly inserted element.

Exceptions

<i>std::invalid_argument</i>	if pos is invalid or end iterator.
------------------------------	------------------------------------

3.1.3.13 operator!=(())

```
template<typename T >
bool CircularLinkedList< T >::operator!= (
    const CircularLinkedList< T > & other ) const [inline]
```

Inequality comparison operator.

Parameters

<i>other</i>	The list to compare with.
--------------	---------------------------

Returns

True if lists are not equal.

3.1.3.14 operator=()

```
template<typename T >
CircularLinkedList& CircularLinkedList< T >::operator= (
    const CircularLinkedList< T > & other ) [inline]
```

Copy assignment operator. Replaces the contents with a copy of another list.

Parameters

<i>other</i>	The list to copy from.
--------------	------------------------

Returns

Reference to this list.

3.1.3.15 operator==()

```
template<typename T >
bool CircularLinkedList< T >::operator== (
    const CircularLinkedList< T > & other ) const [inline]
```

Equality comparison operator. Checks if two circular lists contain the same elements in the same order, accounting for circular rotation.

Parameters

<i>other</i>	The list to compare with.
--------------	---------------------------

Returns

True if lists are equal.

3.1.3.16 pop_front()

```
template<typename T >
void CircularLinkedList< T >::pop_front ( ) [inline]
```

Removes the first element of the list.

Exceptions

<i>std::out_of_range</i>	if the list is empty.
--------------------------	-----------------------

3.1.3.17 push_front()

```
template<typename T >
void CircularLinkedList< T >::push_front (
    const T & value ) [inline]
```

Inserts a new element at the front of the list.

Parameters

<i>value</i>	Value to insert.
--------------	------------------

3.1.3.18 size()

```
template<typename T >
size_t CircularLinkedList< T >::size ( ) const [inline]
```

Returns the number of elements in the list.

Returns

Size of the list.

3.1.4 Friends And Related Function Documentation**3.1.4.1 const_iterator**

```
template<typename T >
friend class const_iterator [friend]
```

3.1.4.2 iterator

```
template<typename T >
friend class iterator [friend]
```

Forward declaration of iterator classes as friends.

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

- [circular_list.h](#)

3.2 CircularLinkedList< T >::const_iterator Class Reference

Constant forward iterator for [CircularLinkedList](#).

```
#include <circular_list.h>
```

Public Types

- using `iterator_category` = `std::forward_iterator_tag`
- using `difference_type` = `std::ptrdiff_t`
- using `value_type` = `const T`
- using `pointer` = `const T *`
- using `reference` = `const T &`

Public Member Functions

- `const_iterator` (`const Element *elem`, `const Element *head`, `bool end`)
Constructs a `const_iterator`.
- `~const_iterator` ()=default
- `const_iterator` (`const const_iterator &`)=default
- `const_iterator & operator=` (`const const_iterator &`)=default
- `const T & operator*` () const
Dereference operator.
- `const T * operator->` () const
Member access operator.
- `const_iterator & operator++` ()
Pre-increment operator. Advances the iterator to the next element.
- `const_iterator operator++` (int)
Post-increment operator. Advances the iterator but returns the previous state.
- `bool operator==` (`const const_iterator &other`) const
Equality comparison.
- `bool operator!=` (`const const_iterator &other`) const
Inequality comparison.

Friends

- class `CircularLinkedList`

3.2.1 Detailed Description

```
template<typename T>
class CircularLinkedList< T >::const_iterator
```

Constant forward iterator for `CircularLinkedList`.

Provides read-only access to list elements.

3.2.2 Member Typedef Documentation

3.2.2.1 difference_type

```
template<typename T >
using CircularLinkedList< T >::const_iterator::difference_type = std::ptrdiff_t
```

3.2.2.2 iterator_category

```
template<typename T >
using CircularLinkedList< T >::const_iterator::iterator_category = std::forward_iterator_tag
```

3.2.2.3 pointer

```
template<typename T >
using CircularLinkedList< T >::const_iterator::pointer = const T*
```

3.2.2.4 reference

```
template<typename T >
using CircularLinkedList< T >::const_iterator::reference = const T&
```

3.2.2.5 value_type

```
template<typename T >
using CircularLinkedList< T >::const_iterator::value_type = const T
```

3.2.3 Constructor & Destructor Documentation

3.2.3.1 const_iterator() [1/2]

```
template<typename T >
CircularLinkedList< T >::const_iterator::const_iterator (
    const Element * elem,
    const Element * head,
    bool end ) [inline]
```

Constructs a [const_iterator](#).

Parameters

<i>elem</i>	Pointer to current element.
<i>head</i>	Pointer to the head element of the list.
<i>end</i>	Flag indicating if this is the end iterator.

3.2.3.2 ~const_iterator()

```
template<typename T >
CircularLinkedList< T >::const_iterator::~~const_iterator ( ) [default]
```

3.2.3.3 const_iterator() [2/2]

```
template<typename T >
CircularLinkedList< T >::const_iterator::const_iterator (
    const const_iterator & ) [default]
```

3.2.4 Member Function Documentation**3.2.4.1 operator!=(())**

```
template<typename T >
bool CircularLinkedList< T >::const_iterator::operator!= (
    const const_iterator & other ) const [inline]
```

Inequality comparison.

Parameters

<i>other</i>	Iterator to compare with.
--------------	---------------------------

Returns

True if iterators differ.

3.2.4.2 operator*()

```
template<typename T >
const T& CircularLinkedList< T >::const_iterator::operator* ( ) const [inline]
```

Dereference operator.

Returns

Const reference to the value of the current element.

Exceptions

<code>std::out_of_range</code>	if iterator is invalid or at end.
--------------------------------	-----------------------------------

3.2.4.3 operator++() [1/2]

```
template<typename T >
const_iterator& CircularLinkedList< T >::const_iterator::operator++ ( ) [inline]
```

Pre-increment operator. Advances the iterator to the next element.

Returns

Reference to the incremented iterator.

Exceptions

<code>std::out_of_range</code>	if iterator is invalid or at end.
--------------------------------	-----------------------------------

3.2.4.4 operator++() [2/2]

```
template<typename T >
const_iterator CircularLinkedList< T >::const_iterator::operator++ (
    int ) [inline]
```

Post-increment operator. Advances the iterator but returns the previous state.

Returns

Iterator before increment.

Exceptions

<code>std::out_of_range</code>	if iterator is invalid or at end.
--------------------------------	-----------------------------------

3.2.4.5 operator->()

```
template<typename T >
const T* CircularLinkedList< T >::const_iterator::operator-> ( ) const [inline]
```

Member access operator.

Returns

Const pointer to the value of the current element.

Exceptions

<code>std::out_of_range</code>	if iterator is invalid or at end.
--------------------------------	-----------------------------------

3.2.4.6 operator=()

```
template<typename T >
const_iterator& CircularLinkedList< T >::const_iterator::operator= (
    const const_iterator & ) [default]
```

3.2.4.7 operator==()

```
template<typename T >
bool CircularLinkedList< T >::const_iterator::operator== (
    const const_iterator & other ) const [inline]
```

Equality comparison.

Parameters

<i>other</i>	Iterator to compare with.
--------------	---------------------------

Returns

True if both iterators point to the same element and end state.

3.2.5 Friends And Related Function Documentation

3.2.5.1 CircularLinkedList

```
template<typename T >
friend class CircularLinkedList [friend]
```

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

- [circular_list.h](#)

3.3 CircularLinkedList< T >::iterator Class Reference

Forward iterator for [CircularLinkedList](#).

```
#include <circular_list.h>
```

Public Types

- using [iterator_category](#) = std::forward_iterator_tag
- using [difference_type](#) = std::ptrdiff_t
- using [value_type](#) = T
- using [pointer](#) = T *
- using [reference](#) = T &

Public Member Functions

- [iterator](#) (Element *elem, Element *head, bool [end](#))
Constructs an iterator.
- [~iterator](#) ()=default
- [iterator](#) (const [iterator](#) &)=default
- [iterator](#) & [operator=](#) (const [iterator](#) &)=default
- T & [operator*](#) () const
Dereference operator.
- T * [operator->](#) () const
Member access operator.
- [iterator](#) & [operator++](#) ()
Pre-increment operator. Advances the iterator to the next element.
- [iterator](#) [operator++](#) (int)
Post-increment operator. Advances the iterator but returns the previous state.
- bool [operator==](#) (const [iterator](#) &other) const
Equality comparison.
- bool [operator!=](#) (const [iterator](#) &other) const
Inequality comparison.

Friends

- class [CircularLinkedList](#)

3.3.1 Detailed Description

```
template<typename T>
class CircularLinkedList< T >::iterator
```

Forward iterator for [CircularLinkedList](#).

Supports traversal of the list from head to end (one past last element).

3.3.2 Member Typedef Documentation

3.3.2.1 difference_type

```
template<typename T >
using CircularLinkedList< T >::iterator::difference_type = std::ptrdiff_t
```

3.3.2.2 iterator_category

```
template<typename T >
using CircularLinkedList< T >::iterator::iterator_category = std::forward_iterator_tag
```

3.3.2.3 pointer

```
template<typename T >
using CircularLinkedList< T >::iterator::pointer = T*
```

3.3.2.4 reference

```
template<typename T >
using CircularLinkedList< T >::iterator::reference = T&
```

3.3.2.5 value_type

```
template<typename T >
using CircularLinkedList< T >::iterator::value_type = T
```

3.3.3 Constructor & Destructor Documentation

3.3.3.1 iterator() [1/2]

```
template<typename T >
CircularLinkedList< T >::iterator::iterator (
    Element * elem,
    Element * head,
    bool end ) [inline]
```

Constructs an iterator.

Parameters

<i>elem</i>	Pointer to current element.
<i>head</i>	Pointer to the head element of the list.
<i>end</i>	Flag indicating if this is the end iterator.

3.3.3.2 ~iterator()

```
template<typename T >
CircularLinkedList< T >::iterator::~iterator ( ) [default]
```

3.3.3.3 iterator() [2/2]

```
template<typename T >
CircularLinkedList< T >::iterator::iterator (
    const iterator & ) [default]
```

3.3.4 Member Function Documentation

3.3.4.1 operator!=(())

```
template<typename T >
bool CircularLinkedList< T >::iterator::operator!= (
    const iterator & other ) const [inline]
```

Inequality comparison.

Parameters

<i>other</i>	Iterator to compare with.
--------------	---------------------------

Returns

True if iterators differ.

3.3.4.2 operator*()

```
template<typename T >
T& CircularLinkedList< T >::iterator::operator* ( ) const [inline]
```

Dereference operator.

Returns

Reference to the value of the current element.

Exceptions

<i>std::out_of_range</i>	if iterator is invalid or at end.
--------------------------	-----------------------------------

3.3.4.3 operator++() [1/2]

```
template<typename T >
iterator& CircularLinkedList< T >::iterator::operator++ ( ) [inline]
```

Pre-increment operator. Advances the iterator to the next element.

Returns

Reference to the incremented iterator.

Exceptions

<i>std::out_of_range</i>	if iterator is invalid or at end.
--------------------------	-----------------------------------

3.3.4.4 operator++() [2/2]

```
template<typename T >
```

```
iterator CircularLinkedList< T >::iterator::operator++ (
    int ) [inline]
```

Post-increment operator. Advances the iterator but returns the previous state.

Returns

Iterator before increment.

Exceptions

<code>std::out_of_range</code>	if iterator is invalid or at end.
--------------------------------	-----------------------------------

3.3.4.5 operator->()

```
template<typename T >
T* CircularLinkedList< T >::iterator::operator-> ( ) const [inline]
```

Member access operator.

Returns

Pointer to the value of the current element.

Exceptions

<code>std::out_of_range</code>	if iterator is invalid or at end.
--------------------------------	-----------------------------------

3.3.4.6 operator=()

```
template<typename T >
iterator& CircularLinkedList< T >::iterator::operator= (
    const iterator & ) [default]
```

3.3.4.7 operator==()

```
template<typename T >
bool CircularLinkedList< T >::iterator::operator== (
    const iterator & other ) const [inline]
```

Equality comparison.

Parameters

<i>other</i>	Iterator to compare with.
--------------	---------------------------

Returns

True if both iterators point to the same element and end state.

3.3.5 Friends And Related Function Documentation

3.3.5.1 CircularLinkedList

```
template<typename T >  
friend class CircularLinkedList [friend]
```

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

- [circular_list.h](#)

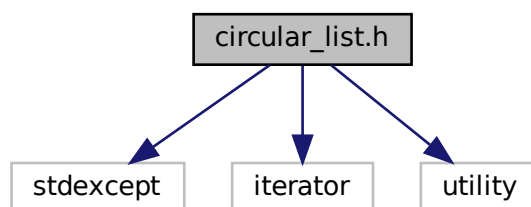
Chapter 4

File Documentation

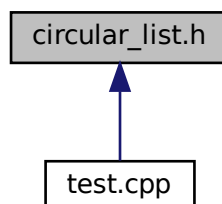
4.1 circular_list.h File Reference

```
#include <stdexcept>
#include <iterator>
#include <utility>
```

Include dependency graph for circular_list.h:



This graph shows which files directly or indirectly include this file:



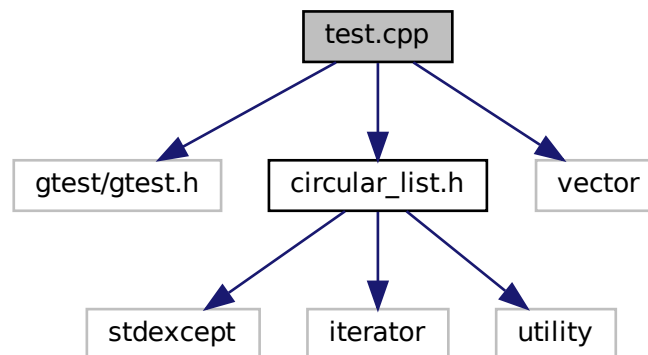
Classes

- class [CircularLinkedList< T >](#)
A circular singly linked list container.
- class [CircularLinkedList< T >::iterator](#)
Forward iterator for [CircularLinkedList](#).
- class [CircularLinkedList< T >::const_iterator](#)
Constant forward iterator for [CircularLinkedList](#).

4.2 test.cpp File Reference

```
#include <gtest/gtest.h>
#include "circular_list.h"
#include <vector>
```

Include dependency graph for test.cpp:



Functions

- [TEST](#) (CircularLinkedListTest, DefaultConstructor)
- [TEST](#) (CircularLinkedListTest, PushFront)
- [TEST](#) (CircularLinkedListTest, PopFront)
- [TEST](#) (CircularLinkedListTest, Iterator)
- [TEST](#) (CircularLinkedListTest, ConstIterator)
- [TEST](#) (CircularLinkedListTest, InsertAfter)
- [TEST](#) (CircularLinkedListTest, EraseAfter)
- [TEST](#) (CircularLinkedListTest, Clear)
- [TEST](#) (CircularLinkedListTest, CopyConstructor)
- [TEST](#) (CircularLinkedListTest, Equality)
- int [main](#) (int argc, char **argv)

4.2.1 Function Documentation

4.2.1.1 main()

```
int main (
    int argc,
    char ** argv )
```

4.2.1.2 TEST() [1/10]

```
TEST (
    CircularLinkedListTest ,
    Clear )
```

4.2.1.3 TEST() [2/10]

```
TEST (
    CircularLinkedListTest ,
    ConstIterator )
```

4.2.1.4 TEST() [3/10]

```
TEST (
    CircularLinkedListTest ,
    CopyConstructor )
```

4.2.1.5 TEST() [4/10]

```
TEST (
    CircularLinkedListTest ,
    DefaultConstructor )
```

4.2.1.6 TEST() [5/10]

```
TEST (
    CircularLinkedListTest ,
    Equality )
```

4.2.1.7 TEST() [6/10]

```
TEST (
    CircularLinkedListTest ,
    EraseAfter )
```

4.2.1.8 TEST() [7/10]

```
TEST (
    CircularLinkedListTest ,
    InsertAfter )
```

4.2.1.9 TEST() [8/10]

```
TEST (
    CircularLinkedListTest ,
    Iterator )
```

4.2.1.10 TEST() [9/10]

```
TEST (
    CircularLinkedListTest ,
    PopFront )
```

4.2.1.11 TEST() [10/10]

```
TEST (
    CircularLinkedListTest ,
    PushFront )
```

Index

- ~CircularLinkedList
 - CircularLinkedList< T >, 7
- ~const_iterator
 - CircularLinkedList< T >::const_iterator, 16
- ~iterator
 - CircularLinkedList< T >::iterator, 21
- begin
 - CircularLinkedList< T >, 7
- cbegin
 - CircularLinkedList< T >, 8
- cend
 - CircularLinkedList< T >, 8
- circular_list.h, 25
- CircularLinkedList
 - CircularLinkedList< T >, 7
 - CircularLinkedList< T >::const_iterator, 18
 - CircularLinkedList< T >::iterator, 24
- CircularLinkedList< T >, 5
 - ~CircularLinkedList, 7
 - begin, 7
 - cbegin, 8
 - cend, 8
 - CircularLinkedList, 7
 - clear, 8
 - const_iterator, 13
 - empty, 8
 - end, 9
 - erase_after, 9
 - front, 10
 - insert_after, 10
 - iterator, 13
 - operator!=, 11
 - operator=, 11
 - operator==, 12
 - pop_front, 12
 - push_front, 12
 - size, 13
- CircularLinkedList< T >::const_iterator, 13
 - ~const_iterator, 16
 - CircularLinkedList, 18
 - const_iterator, 15, 16
 - difference_type, 14
 - iterator_category, 15
 - operator!=, 16
 - operator*, 16
 - operator++, 17
 - operator->, 17
 - operator=, 18
 - operator==, 18
 - pointer, 15
 - reference, 15
 - value_type, 15
- CircularLinkedList< T >::iterator, 19
 - ~iterator, 21
 - CircularLinkedList, 24
 - difference_type, 20
 - iterator, 21
 - iterator_category, 20
 - operator!=, 21
 - operator*, 22
 - operator++, 22
 - operator->, 23
 - operator=, 23
 - operator==, 23
 - pointer, 20
 - reference, 20
 - value_type, 20
- clear
 - CircularLinkedList< T >, 8
- const_iterator
 - CircularLinkedList< T >, 13
 - CircularLinkedList< T >::const_iterator, 15, 16
- difference_type
 - CircularLinkedList< T >::const_iterator, 14
 - CircularLinkedList< T >::iterator, 20
- empty
 - CircularLinkedList< T >, 8
- end
 - CircularLinkedList< T >, 9
- erase_after
 - CircularLinkedList< T >, 9
- front
 - CircularLinkedList< T >, 10
- insert_after
 - CircularLinkedList< T >, 10
- iterator
 - CircularLinkedList< T >, 13
 - CircularLinkedList< T >::iterator, 21
- iterator_category
 - CircularLinkedList< T >::const_iterator, 15
 - CircularLinkedList< T >::iterator, 20
- main
 - test.cpp, 26

operator!=
 CircularLinkedList< T >, [11](#)
 CircularLinkedList< T >::const_iterator, [16](#)
 CircularLinkedList< T >::iterator, [21](#)

operator*
 CircularLinkedList< T >::const_iterator, [16](#)
 CircularLinkedList< T >::iterator, [22](#)

operator++
 CircularLinkedList< T >::const_iterator, [17](#)
 CircularLinkedList< T >::iterator, [22](#)

operator->
 CircularLinkedList< T >::const_iterator, [17](#)
 CircularLinkedList< T >::iterator, [23](#)

operator=
 CircularLinkedList< T >, [11](#)
 CircularLinkedList< T >::const_iterator, [18](#)
 CircularLinkedList< T >::iterator, [23](#)

operator==
 CircularLinkedList< T >, [12](#)
 CircularLinkedList< T >::const_iterator, [18](#)
 CircularLinkedList< T >::iterator, [23](#)

pointer
 CircularLinkedList< T >::const_iterator, [15](#)
 CircularLinkedList< T >::iterator, [20](#)

pop_front
 CircularLinkedList< T >, [12](#)

push_front
 CircularLinkedList< T >, [12](#)

reference
 CircularLinkedList< T >::const_iterator, [15](#)
 CircularLinkedList< T >::iterator, [20](#)

size
 CircularLinkedList< T >, [13](#)

TEST
 test.cpp, [27](#), [28](#)

test.cpp, [26](#)
 main, [26](#)
 TEST, [27](#), [28](#)

value_type
 CircularLinkedList< T >::const_iterator, [15](#)
 CircularLinkedList< T >::iterator, [20](#)