BagContainerAdapter

Generated by Doxygen 1.9.1

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Module Documentation

4.1 Initializations for different container types.

4.1.1 Detailed Description

4.2 Functions called by destructor for different underlying container types.

4.2.1 Detailed Description

4.3 Insert functionality for various underlying container types.

4.3.1 Detailed Description

4.4 Erase functionality for various underlying container types.

4.4.1 Detailed Description

4.5 Functionality for getting the first element for various container types.

4.5.1 Detailed Description

4.6 Funcionality for getting the amount of elements for various container types.

4.6.1 Detailed Description

Chapter 5

Class Documentation

5.1 BagContainerAdaptor< Container > Class Template Reference

`#include <bag_container_adaptor.hpp>`

Public Types

- using value_type = typename Container::value_type

  The value type of the underlying container.
- using iterator = typename Container::iterator

  The iterator of the underlying container.
- using const_iterator = typename Container::const_iterator

  The constant iterator of the underlying container.

Public Member Functions

- BagContainerAdaptor () noexcept
- ~BagContainerAdaptor () noexcept
- BagContainerAdaptor (Container &&container) noexcept
- BagContainerAdaptor & operator= (BagContainerAdaptor &&other) noexcept
- template<typename OtherContainer , typename = std::enable_if<std::is_same<Container, OtherContainer>::value>>
  BagContainerAdaptor (BagContainerAdaptor< OtherContainer > &&other) noexcept
- BagContainerAdaptor (const BagContainerAdaptor &other)=delete
- BagContainerAdaptor & operator= (const BagContainerAdaptor &other)=delete
- template<typename Iterator >
  iterator insert (Iterator pos, const value_type &value)
- iterator insert (const value_type &value)
- iterator erase (iterator pos)
- iterator erase (const value_type &value)
- template<typename FirstType , typename LastType >
  iterator erase (FirstType first, LastType last)
- void swap (BagContainerAdaptor &other) noexcept
- iterator begin () noexcept
- iterator end () noexcept
- const_iterator cbegin () const noexcept
- const_iterator cend () const noexcept

- iterator find (const value_type &value) noexcept
- const_iterator find (const value_type &value) const noexcept
- value_type & front () noexcept
- const value_type & front () const noexcept
- value_type & back () noexcept
- const value_type & back () const noexcept
- size_t size () const noexcept
- bool empty () const noexcept

5.1.1 Detailed Description

template<typename Container>
class BagContainerAdaptor< Container >

Bag is an abstract data type that can store a collection of elements without regard to their order. Equal elements can appear multiple times in a bag. Although the elements container in a bag have no inherit order, iterating over the bag elements is guaranteed to visit each element exactly once. This bag takes in an stl container as a template argument and provides the same functionality for each container type following the design pattern of an adapter.

Template Parameters

Container	The underlying container type.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 BagContainerAdaptor() [1/4]

```
template<typename Container >
BagContainerAdaptor< Container >::BagContainerAdaptor ( )  [inline], [noexcept]
```

Constructor.

Postcondition

The `BagContainerAdaptor` object is constructed, and the `m_container` is initialized to an empty state.

Exceptions

noexcept	The constructor is marked `noexcept` to guarantee no exceptions will be thrown during the construction, providing a strong exception safety guarantee.

5.1.2.2 ~BagContainerAdaptor()

```
template<typename Container >
BagContainerAdaptor< Container >::~BagContainerAdaptor ( )  [inline], [noexcept]
```

Destructor.

Postcondition

The BagContainerAdaptor object is destructed, and the m_container is deallocated, releasing any resources held by the container.

Exceptions

noexcept	The destructor is marked noexcept to guarantee no exceptions will be thrown during the destruction, providing a strong exception safety guarantee.

5.1.2.3 BagContainerAdaptor() [2/4]

```
template<typename Container >
BagContainerAdaptor< Container >::BagContainerAdaptor (
            Container && container )  [inline], [noexcept]
```

Move constructor.

Parameters

container	The underlying container from which the BagContainerAdaptor is constructed.

Precondition

The underlying container must have a move constructor to support moving its contents.

Postcondition

The BagContainerAdapter is constructed, taking ownership of the contents of the underlying container.

Exceptions

noexcept	The move constructor is marked noexcept to guarantee no exceptions will be thrown during the move operation, providing strong exception safety.

5.1.2.4 BagContainerAdaptor() [3/4]

```
template<typename Container >
```

```
template<typename OtherContainer , typename = std::enable_if<std::is_same<Container, Other↩
Container>::value>>
BagContainerAdaptor< Container >::BagContainerAdaptor (
            BagContainerAdaptor< OtherContainer > && other )  [inline], [noexcept]
```

Forwarding constructor for self-instantiation.

Parameters

other	The other `BagContainerAdaptor` from which we are initializing from.

Template Parameters

OtherContainer	The template argument for the other container.

Precondition

The `OtherContainer` type must be the same as the `Container` type to ensure valid instantiation.

Postcondition

The current `BagContainerAdaptor` object takes ownership of the contents of the other 'BagContainerAdaptor'. The other `BagContainerAdaptor` is left in a valid but unspecified state.

Exceptions

noexcept	The forwarding constructor is marked noexcept to guarantee no exceptions will be thrown during the move operation, providing a strong exception safety guarantee.

5.1.2.5 BagContainerAdaptor() [4/4]

```
template<typename Container >
BagContainerAdaptor< Container >::BagContainerAdaptor (
            const BagContainerAdaptor< Container > & other )  [delete]
```

Copy constructor.

Parameters

other	The other BagContainerAdaptor where we copy from.

Note

This copy constructor has been explicitly deleted to prevent copying.

5.1.3 Member Function Documentation

5.1.3.1 back() [1/2]

```
template<typename Container >
const value_type& BagContainerAdaptor< Container >::back ( ) const  [inline], [noexcept]
```

Get reference to the last element in the underlying container.

Returns

Reference to the last element in the underlying container.

Precondition

The container must not be empty.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.1.3.2 back() [2/2]

```
template<typename Container >
value_type& BagContainerAdaptor< Container >::back ( )  [inline], [noexcept]
```

Get reference to the last element in the underlying container.

Returns

Reference to the last element in the underlying container.

Precondition

The container must not be empty.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.1.3.3 begin()

```
template<typename Container >
iterator BagContainerAdaptor< Container >::begin ( )  [inline], [noexcept]
```

Get iterator pointing to the first element in the underlying container.

Returns

Iterator pointing to the first element in the underlying container.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.1.3.4 cbegin()

```
template<typename Container >
const_iterator BagContainerAdaptor< Container >::cbegin ( ) const  [inline], [noexcept]
```

Get constant iterator pointing to the first element in the underlying container.

Returns

Constant iterator pointing to the first element in the underlying container.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.1.3.5 cend()

```
template<typename Container >
const_iterator BagContainerAdaptor< Container >::cend ( ) const  [inline], [noexcept]
```

Get constant iterator pointing to the last element in the underlying container.

Returns

Contant iterator pointing to the last element in the underlying container.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.1.3.6 empty()

```
template<typename Container >
bool BagContainerAdaptor< Container >::empty ( ) const  [inline], [noexcept]
```

Get boolean describing if the underlying container is empty or not.

Returns

Boolean describing if the underlying container is empty or not.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.1.3.7 end()

```
template<typename Container >
iterator BagContainerAdaptor< Container >::end ( )  [inline], [noexcept]
```

Get iterator pointing to the last element in the underlying container.

Returns

Iterator pointing to the last element in the underlying container.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.1.3.8 erase() [1/3]

```
template<typename Container >
iterator BagContainerAdaptor< Container >::erase (
            const value_type & value )  [inline]
```

Erase all elements that have the specified value in the underlying container.

Parameters

value	The value of the elements that are removed.

Returns

An iterator following the last removed element.

Postcondition

All elements equal to the specified value in the underlying container are removed, and the BagContainerAdaptor object is modified accordingly.

Exceptions

Depending	on the underlying container's erase operation, this function might throw exceptions like: • For std::vector with move iterator: std::out_of_range if the pos iterator is invalid or if the move constructor of the contained type throws an exception. • For other containers: No exceptions are thrown by the erase operation itself unless specified by the container.

Note

The iterator returned points to the element that follows the last removed element in the underlying container. If no element with the specified value is found or if all occurrences of the value are removed, the returned iterator is the end() iterator of the container.

5.1.3.9 erase() [2/3]

```
template<typename Container >
template<typename FirstType , typename LastType >
iterator BagContainerAdaptor< Container >::erase (
            FirstType first,
            LastType last )  [inline]
```

Erase all elements between two iterators from the underlying container.

Parameters

first	The first element in the range of elements being removed.
last	The last element of the range of the elements being removed.

Template Parameters

FirstType	The type of iterator for the first element in the range.
LastType	The type of iterator for the last element in the range.

Returns

An iterator following the last removed element.

Postcondition

All elements between the specified range [first, last] in the underlying container are removed, and the `BagContainerAdaptor` object is modified accordingly.

Exceptions

Depending	on the underlying container's erase operation, this function might throw exceptions like: • For std::vector with move iterator: std::out_of_range if the iterators are invalid or if the move constructor of the contained type throws an exception. • For other containers: No exceptions are thrown by the erase operation itself unless specified by the container.

Note

The iterator returned points to the element that follows the last removed element in the underlying container. If no elements are removed, the returned iterator is the one pointed to by the "last" iterator parameter.

5.1.3.10 erase() [3/3]

```
template<typename Container >
iterator BagContainerAdaptor< Container >::erase (
            iterator pos )  [inline]
```

Erase element from the specified position in the underlying container.

Parameters

pos	The specified position where the element is erased.

Returns

An iterator following the last removed element.

Precondition

The `pos` iterator must be a valid iterator that points to a position within the underlying container.

Postcondition

The element at the specified `pos` in the underlying container is removed, and the `BagContainerAdaptor` object is modified accordingly.

Exceptions

Depending	on the underlying container's erase operation, this function might throw exceptions like: • For std::vector: std::out_of_range if the pos iterator is invalid.

Note

The iterator returned points to the element that follows the erased element in the underlying container. If `pos` points to the last element, the returned iterator is the `end()` iterator of the container.

5.1.3.11 find() [1/2]

```
template<typename Container >
const_iterator BagContainerAdaptor< Container >::find (
            const value_type & value ) const  [inline], [noexcept]
```

Get constant iterator pointing to the first instance of element with specified value in const context.

Parameters

value	The value to compare elements to.

Returns

An iterator pointing to the first instance of element with specified value.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.1.3.12 find() [2/2]

```
template<typename Container >
iterator BagContainerAdaptor< Container >::find (
            const value_type & value )  [inline], [noexcept]
```

Get iterator pointing to the first instance of element with specified value.

Parameters

value	The value to compare elements to.

Returns

An iterator pointing to the first instance of element with specified value.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.1.3.13 front() [1/2]

```
template<typename Container >
const value_type& BagContainerAdaptor< Container >::front ( ) const  [inline], [noexcept]
```

Get reference to the first element in the underlying container in const context.

Returns

Reference to the first element in the underlying container.

Precondition

The container must not be empty.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.1.3.14 front() [2/2]

```
template<typename Container >
value_type& BagContainerAdaptor< Container >::front ( )  [inline], [noexcept]
```

Get reference to the first element in the underlying container.

Returns

Reference to the first element in the underlying container.

Precondition

The container must not be empty.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.1.3.15 insert() [1/2]

```
template<typename Container >
iterator BagContainerAdaptor< Container >::insert (
            const value_type & value )  [inline]
```

Insert element to the last position of the underlying container.

Parameters

value	The value to be inserted.

Returns

An iterator that points to the inserted element.

Postcondition

The `value` is inserted at the last position of the underlying container, and the `BagContainerAdaptor` object is modified accordingly.

Exceptions

Depending	on the underlying container's insertion operations, this function might throw exceptions like `std::bad_alloc` if memory allocation fails.

5.1.3.16 insert() [2/2]

```
template<typename Container >
template<typename Iterator >
iterator BagContainerAdaptor< Container >::insert (
            Iterator pos,
            const value_type & value )  [inline]
```

Insert element to specified position of the underlying container.

Parameters

pos	The specified position where the element is inserted.
value	The value to be inserted.

Template Parameters

Iterator	The iterator type of the underlying container.

Returns

An iterator that points to the inserted element.

Precondition

The `pos` iterator must be a valid iterator that points to a position within the underlying container.

Postcondition

The `value` is inserted at the specified `pos` in the underlying container, and the `BagContainerAdaptor` object is modified accordingly.

Exceptions

Depending	on the underlying container's insertion operations, this function might throw exceptions like `std::bad_alloc` if memory allocation fails.

5.1.3.17 operator=() [1/2]

```
template<typename Container >
BagContainerAdaptor& BagContainerAdaptor< Container >::operator= (
            BagContainerAdaptor< Container > && other )  [inline], [noexcept]
```

Parameters

other	The underlying container from which the `BagContainerAdaptor` is created.

Returns

Reference to the current `BagContainerAdaptor` object after the move assignment.

Precondition

The `Container` type must have a move assignment operator to support moving its contents.

Postcondition

The current `BagContainerAdaptor` object takes ownership of the contents of the other `BagContainerAdaptor`. The other `BagContainerAdaptor` is left in a valid but unspecified state.

Exceptions

noexcept	The move assignment operator is marked noexcept to guarantee no exceptions will be thrown during the move assignment, providing a string exception safety.

5.1.3.18 operator=() [2/2]

```
template<typename Container >
BagContainerAdaptor& BagContainerAdaptor< Container >::operator= (
            const BagContainerAdaptor< Container > & other )  [delete]
```

Copy assignment operator.

Parameters

other	The other BagContainerAdaptor where we copy from.

Note

This copy assignment operator has been explicitly deleted to prevent copying.

5.1.3.19 size()

```
template<typename Container >
size_t BagContainerAdaptor< Container >::size ( ) const  [inline], [noexcept]
```

Get the amount of elements in the underlying container.

Returns

The amount of elements in the underlying container.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.1.3.20 swap()

```
template<typename Container >
void BagContainerAdaptor< Container >::swap (
            BagContainerAdaptor< Container > & other )  [inline], [noexcept]
```

Swap the contents of two BagContainerAdaptors.

Parameters

other	The other bag to be swapped with.

Postcondition

The contents of this BagContainerAdaptor are swapped with the contents of the "other" BagContainerAdaptor.

Exceptions

noexcept	No exceptions are thrown by this operation.

Note

The swap operation is performed using the underlying container's swap operation, which is noexcept for most standard containers (like std::vector, std::deque, std::list, std::forward_list, std::multiset, and std::unordered_multiset), ensuring a fast and exception-safe swap.

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

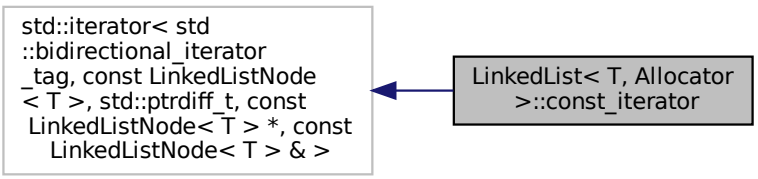
- BagContainerAdaptor/include/BagContainerAdaptor/bag_container_adaptor.hpp

5.2 LinkedList< T, Allocator >::const_iterator Class Reference

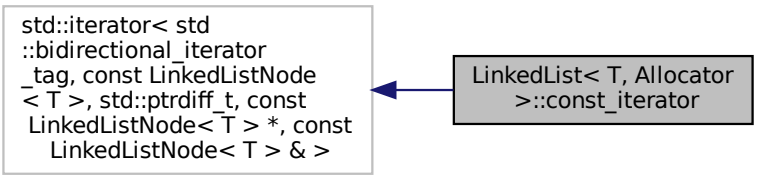
A bidirectional constant iterator for traversing elements in the linked list.

`#include <linked_list.hpp>`

Inheritance diagram for LinkedList< T, Allocator >::const_iterator:



Collaboration diagram for LinkedList< T, Allocator >::const_iterator:



Public Member Functions

- const_iterator () noexcept
  Default constructor.
- const_iterator (LinkedListNode< T > *node) noexcept
- const T & operator* () const noexcept
- const T * operator-> () const noexcept
- const_iterator & operator++ () noexcept
- const_iterator operator++ (int) noexcept
- const_iterator & operator-- ()
- const_iterator operator-- (int)
- bool operator== (const const_iterator &other) const noexcept
- bool operator!= (const const_iterator &other) const noexcept
- const_iterator (const typename LinkedList< T >::iterator &it) noexcept
- const_iterator & operator= (const typename LinkedList< T >::iterator &it) noexcept
- const_iterator (typename LinkedList< T >::iterator &&it) noexcept
- const_iterator & operator= (typename LinkedList< T >::iterator &&it) noexcept
- const_iterator (const const_iterator &other) noexcept=default
- const_iterator & operator= (const const_iterator &other) noexcept=default
- const_iterator (const_iterator &&other) noexcept=default
- const_iterator & operator= (const_iterator &&other) noexcept=default
- const LinkedListNode< T > * getNode () const noexcept

5.2.1 Detailed Description

template<typename T, typename Allocator = std::allocator<LinkedListNode<T>>>
class LinkedList< T, Allocator >::const_iterator

A bidirectional constant iterator for traversing elements in the linked list.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 const_iterator() [1/5]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::const_iterator::const_iterator (
            LinkedListNode< T > * node )  [inline], [explicit], [noexcept]
```

Constructor.

Parameters

node	Pointer to the LinkedListNode to initialize constant iterator with.

Postcondition

The iterator is constructed with the given LinkedListNode as the current node.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.2.2 const_iterator() [2/5]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::const_iterator::const_iterator (
            const typename LinkedList< T >::iterator & it )  [inline], [noexcept]
```

Copy constructibility from non-const iterator.

Parameters

it	The `iterator` to copy construct from.

Postcondition

The constant iterator is constructed with the same current node as the `iterator`.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.2.3 const_iterator() [3/5]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::const_iterator::const_iterator (
            typename LinkedList< T >::iterator && it )  [inline], [noexcept]
```

Move constructibility from non-const iterator.

Parameters

it	The non-const iterator to be moved from.

Postcondition

The const_iterator is constructed, taking ownership of the internal pointer from the non-const iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.2.4 const_iterator() [4/5]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::const_iterator::const_iterator (
            const const_iterator & other )  [default], [noexcept]
```

Copy constructor.

Parameters

other	The constant iterator to be copied.

Postcondition

The iterator is constructed as a copy of the other constant iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.2.5 const_iterator() [5/5]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::const_iterator::const_iterator (
            const_iterator && other )  [default], [noexcept]
```

Move constructor.

Parameters

other	The constant iterator to be moved.

Postcondition

The constant iterator is constructed by moving the other iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.3 Member Function Documentation

5.2.3.1 getNode()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const LinkedListNode<T>* LinkedList< T, Allocator >::const_iterator::getNode ( ) const  [inline],
[noexcept]
```

Get the Node where the constant iterator is pointing.

Returns

A pointer to the current node where the constant iterator is pointing.

Postcondition

Returns a pointer to the current node where the constant iterator is pointing.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.3.2 operator"!=()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
bool LinkedList< T, Allocator >::const_iterator::operator!= (
            const const_iterator & other ) const  [inline], [noexcept]
```

Inequality comparison operator for the iterator.

Parameters

other	The constant iterator to compare with.

Returns

True if both iterators do not point to the same node, otherwise false.

Postcondition

Checks if the current node of this constant iterator is not equal to the current node of the other constant iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.3.3 operator*()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const T& LinkedList< T, Allocator >::const_iterator::operator* ( ) const  [inline], [noexcept]
```

Deference operator for the constant iterator.

Returns

A constant reference to the data of the current node.

Postcondition

Returns a constant reference to the data of the current node.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.3.4 operator++() [1/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_iterator& LinkedList< T, Allocator >::const_iterator::operator++ ( )  [inline], [noexcept]
```

Pre-increment operator for the constant iterator.

Returns

A reference to the constant iterator after the increment.

Postcondition

Moves the constant iterator to the next node in the linked list.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.3.5 operator++() [2/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_iterator LinkedList< T, Allocator >::const_iterator::operator++ (
            int  )  [inline], [noexcept]
```

Post-increment operator for the constant iterator.

Returns

An iterator pointing to the previous position before the increment.

Postcondition

Moves the constant iterator to the next node in the linked list.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.3.6 operator--() [1/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_iterator& LinkedList< T, Allocator >::const_iterator::operator-- ( )  [inline]
```

Pre-decrement operator for the constant iterator.

Returns

A reference to the constant iterator after the decrement.

Postcondition

Moves the iterator to the previous node in the linked list.

5.2.3.7 operator--() [2/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_iterator LinkedList< T, Allocator >::const_iterator::operator-- (
            int  )  [inline]
```

Post-decrement operator for the constant iterator.

Returns

An iterator pointing to the previous position before the decrement (this).

Postcondition

Moves the iterator to the previous node in the linked list.

5.2.3.8 operator->()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const T* LinkedList< T, Allocator >::const_iterator::operator-> ( ) const  [inline], [noexcept]
```

Arrow operator for the constant iterator.

Returns

A constant pointer to the data of the current node.

Postcondition

Returns a constant pointer to the data of the current node.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.3.9 operator=() [1/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_iterator& LinkedList< T, Allocator >::const_iterator::operator= (
            const const_iterator & other )  [default], [noexcept]
```

Copy assignment operator.

Parameters

other	The constant iterator to be copied.

Returns

A reference to the constant iterator after the assignment.

Postcondition

The constant iterator as assigned as a copy of the other constant iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.3.10 operator=() [2/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_iterator& LinkedList< T, Allocator >::const_iterator::operator= (
            const typename LinkedList< T >::iterator & it )  [inline], [noexcept]
```

Copy assignment from non-const iterator.

Parameters

it	The `iterator` to copy assign from.

Returns

A reference to the constant iterator after the assignment.

Postcondition

The constant iterator is assigned with the same current node as the `iterator`.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.3.11 operator=() [3/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
```

```
const_iterator& LinkedList< T, Allocator >::const_iterator::operator= (
            const_iterator && other )  [default], [noexcept]
```

Move assignment operator.

Parameters

other	The constant iterator to be moved.

Returns

A reference to the constant iterator after the assignment.

Postcondition

The constant iterator is assigned by moving the other constant iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.3.12 operator=() [4/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_iterator& LinkedList< T, Allocator >::const_iterator::operator= (
            typename LinkedList< T >::iterator && it )  [inline], [noexcept]
```

Move assingment from non-const iterator.

Parameters

it	The non-const iterator to be moved from.

Returns

A reference to the constant iterator after the assignment.

Postcondition

The const_iterator is assigned the value of the non-const iterator, taking ownership of the internal pointer.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.2.3.13 operator==()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
bool LinkedList< T, Allocator >::const_iterator::operator== (
            const const_iterator & other ) const  [inline], [noexcept]
```

Equality comparison operator for the iterator.

Parameters

other	The constant iterator to compare with.

Returns

True if both of the constant iterators point to the same node, otherwise false.

Postcondition

Checks if the current node of this constant iterator is equal to the current node of the other iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

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

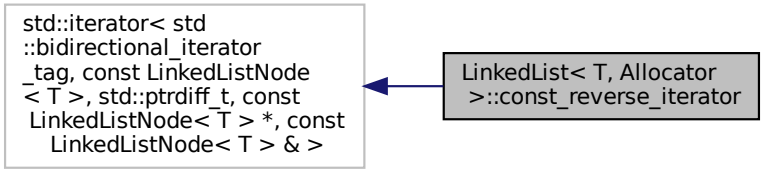
- BagContainerAdaptor/include/BagContainerAdaptor/linked_list.hpp

5.3 LinkedList< T, Allocator >::const_reverse_iterator Class Reference

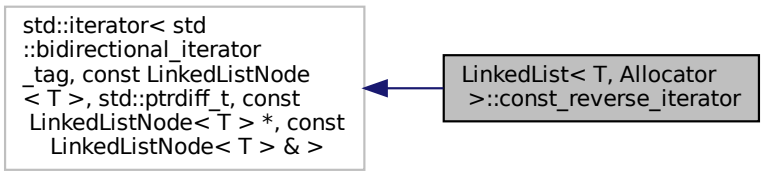
A bidirectional constant reverse iterator for traversing items backwards in the linked list.

`#include <linked_list.hpp>`

Inheritance diagram for LinkedList< T, Allocator >::const_reverse_iterator:



Collaboration diagram for LinkedList< T, Allocator >::const_reverse_iterator:



Public Member Functions

- const_reverse_iterator () noexcept
- const_reverse_iterator (LinkedListNode< T > *node) noexcept
- const T & operator* () const noexcept
- const T * operator-> () const noexcept
- const_reverse_iterator & operator++ ()
- const_reverse_iterator operator++ (int)
- const_reverse_iterator & operator-- ()
- const_reverse_iterator operator-- (int) noexcept
- bool operator== (const const_reverse_iterator &other) const noexcept
- bool operator!= (const const_reverse_iterator &other) const noexcept
- const_reverse_iterator (const typename LinkedList< T >::reverse_iterator &it) noexcept
- const_reverse_iterator & operator= (const typename LinkedList< T >::reverse_iterator &it) noexcept
- const_reverse_iterator (typename LinkedList< T >::reverse_iterator &&it) noexcept
- const_reverse_iterator & operator= (typename LinkedList< T >::reverse_iterator &&it) noexcept
- const_reverse_iterator (const const_reverse_iterator &other) noexcept=default
- const_reverse_iterator & operator= (const const_reverse_iterator &other) noexcept=default
- const_reverse_iterator (const_reverse_iterator &&other) noexcept=default
- const_reverse_iterator & operator= (const_reverse_iterator &&other) noexcept=default
- LinkedListNode< T > * getNode () const noexcept

5.3.1 Detailed Description

template<typename T, typename Allocator = std::allocator<LinkedListNode<T>>>
class LinkedList< T, Allocator >::const_reverse_iterator

A bidirectional constant reverse iterator for traversing items backwards in the linked list.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 const_reverse_iterator() [1/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::const_reverse_iterator::const_reverse_iterator ( )  [inline],
[noexcept]
```

Default constructor.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.2.2 const_reverse_iterator() [2/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::const_reverse_iterator::const_reverse_iterator (
            LinkedListNode< T > * node )  [inline], [explicit], [noexcept]
```

Constructor.

Parameters

node	Pointer to the `LinkedListNode` to initialize constant reverse iterator with.

Postcondition

The constant reverse iterator is constructed with the given `LinkedListNode` as the current node.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.2.3 const_reverse_iterator() [3/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::const_reverse_iterator::const_reverse_iterator (
            const typename LinkedList< T >::reverse_iterator & it )  [inline], [noexcept]
```

Copy constructibility from non-const reverse iterator.

Parameters

it	The reverse iterator to copy construct from.

Postcondition

The constant reverse iterator is constructed with the same current node as the non-const reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.2.4 const_reverse_iterator() [4/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::const_reverse_iterator::const_reverse_iterator (
            typename LinkedList< T >::reverse_iterator && it )  [inline], [noexcept]
```

Move constructibility from non-const reverse iterator.

Parameters

it	The reverse iterator to be moved from.

Postcondition

The constant reverse iterator is constructed, taking ownership of the internal pointer from the non-const reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.2.5 const_reverse_iterator() [5/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::const_reverse_iterator::const_reverse_iterator (
            const const_reverse_iterator & other )  [default], [noexcept]
```

Copy constructor.

Parameters

other	The constant reverse iterator to be copied.

Postcondition

The constant reverse iterator is constructed as a copy of the other constant reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.2.6 const_reverse_iterator() [6/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::const_reverse_iterator::const_reverse_iterator (
            const_reverse_iterator && other )  [default], [noexcept]
```

Move constructor

Parameters

other	The constant reverse iterator to be moved.

Postcondition

The constant reverse iterator is constructed by moving the other constant reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.3 Member Function Documentation

5.3.3.1 getNode()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedListNode<T>* LinkedList< T, Allocator >::const_reverse_iterator::getNode ( ) const
[inline], [noexcept]
```

Get the node where the constant reverse iterator is pointing.

Returns

A pointer to the current node where the constant reverse iterator is pointing.

Postcondition

Returns a pointer to the current node where the iterator is pointing.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.3.2 operator"!=()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
bool LinkedList< T, Allocator >::const_reverse_iterator::operator!= (
            const const_reverse_iterator & other ) const  [inline], [noexcept]
```

Inequality comparison operator for the constant reverse iterator.

Parameters

other	The constant reverse iterator to compare with.

Returns

True of both constant reverse iterators do not point to the same node, otherwise false.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.3.3 operator*()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const T& LinkedList< T, Allocator >::const_reverse_iterator::operator* ( ) const  [inline],
[noexcept]
```

Dereference operator for the constant reverse iterator.

Returns

A constant reference to the data of the current node.

Postcondition

Returns a constant reference to the data of the current node.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.3.4 operator++() [1/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_reverse_iterator& LinkedList< T, Allocator >::const_reverse_iterator::operator++ ( )
[inline]
```

Pre-increment operator for the constant reverse iterator.

Returns

A reference to the constant reverse iterator after the increment.

Postcondition

Moves the constant reverse iterator to the previous node in the linked list.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.3.5 operator++() [2/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_reverse_iterator LinkedList< T, Allocator >::const_reverse_iterator::operator++ (
            int  )  [inline]
```

Post-increment operator for the constant reverse iterator.

Returns

A constant reverse iterator pointing to the previous position before the increment.

Postcondition

Moves the constant reverse iterator to the previous node in the linked list.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.3.6 operator--() [1/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_reverse_iterator& LinkedList< T, Allocator >::const_reverse_iterator::operator-- ( )
[inline]
```

Pre-decrement operator for the constant iterator.

Returns

A reference to the constant reverse iterator after the decrement.

Postcondition

Moves the constant reverse iterator to the next node in the linked list.

5.3.3.7 operator--() [2/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_reverse_iterator LinkedList< T, Allocator >::const_reverse_iterator::operator-- (
            int  )  [inline], [noexcept]
```

Post-decrement operator for the constant reverse iterator.

Returns

A constant reverse iterator pointing to the previous position before the decrement.

Postcondition

Moves the constant reverse iterator to the next node in the linked list.

5.3.3.8 operator->()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const T* LinkedList< T, Allocator >::const_reverse_iterator::operator-> ( ) const  [inline],
[noexcept]
```

Arrow operator for the constant reverse iterator.

Returns

A constant pointer to data of the current node.

Postcondition

Returns a constant pointer to the data of the current node.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.3.9 operator=() [1/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_reverse_iterator& LinkedList< T, Allocator >::const_reverse_iterator::operator= (
            const const_reverse_iterator & other )  [default], [noexcept]
```

Copy assignment operator.

Parameters

other	The constant reverse iterator to be copied.

Returns

A reference to the constant reverse iterator after the assignment.

Postcondition

The constant reverse iterator is assigned as a copy of the other constant reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.3.10 operator=() [2/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_reverse_iterator& LinkedList< T, Allocator >::const_reverse_iterator::operator= (
            const typename LinkedList< T >::reverse_iterator & it )  [inline], [noexcept]
```

Copy assignment from non-const reverse iterator.

Parameters

it	The reverse iterator to copy assign from.

Returns

A reference to the constant reverse iterator after the assignment.

Postcondition

The constant reverse iterator is assigned with the same current node as the non-const reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.3.11 operator=() [3/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_reverse_iterator& LinkedList< T, Allocator >::const_reverse_iterator::operator= (
            const_reverse_iterator && other )  [default], [noexcept]
```

Move assignment operator.

Parameters

other	The constant reverse iterator to be moved.

Returns

A reference to the constant reverse iterator after the assignment.

Postcondition

The constant reverse iterator is assigned by moving the other constant reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.3.12 operator=() [4/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_reverse_iterator& LinkedList< T, Allocator >::const_reverse_iterator::operator= (
            typename LinkedList< T >::reverse_iterator && it )  [inline], [noexcept]
```

Move assignment from non-const reverse iterator.

Parameters

it	The reverse iterator to be moved from.

Returns

A reference to the constant reverse iterator after the assignment.

Postcondition

The constant reverse iterator is assigned with the value of the reverse iterator, taking ownership of the internal pointer.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.3.3.13 operator==()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
bool LinkedList< T, Allocator >::const_reverse_iterator::operator== (
            const const_reverse_iterator & other ) const  [inline], [noexcept]
```

Equality comparison operator for the constant reverse iterator.

Parameters

other	The constant reverse iterator to compare with.

Returns

True if both of the constant reverse iterators point to the same node, otherwise false.

Postcondition

Checks if the current node if this constant reverse iterator is equal to the current of the other iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

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

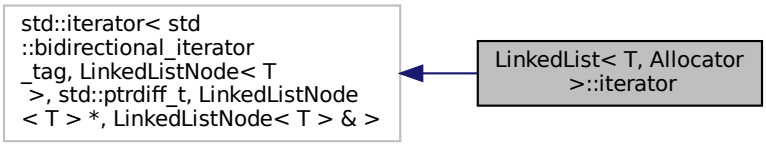
- BagContainerAdaptor/include/BagContainerAdaptor/linked_list.hpp

5.4 LinkedList< T, Allocator >::iterator Class Reference

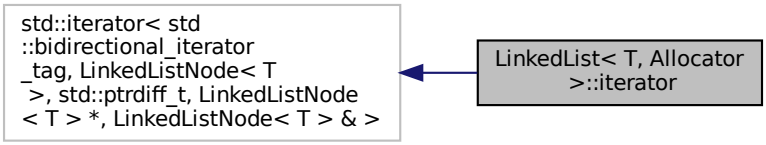
A bidirectional iterator for traversing elements in the linked list.

`#include <linked_list.hpp>`

Inheritance diagram for LinkedList< T, Allocator >::iterator:



Collaboration diagram for LinkedList< T, Allocator >::iterator:



Public Member Functions

- iterator () noexcept
- iterator (LinkedListNode< T > *node) noexcept
- T & operator* () const noexcept
- T * operator-> () const noexcept
- iterator & operator++ () noexcept
- iterator operator++ (int) noexcept
- iterator & operator-- ()
- iterator operator-- (int)
- bool operator== (const iterator &other) const noexcept
- bool operator!= (const iterator &other) const noexcept
- iterator (const typename LinkedList< T >::const_iterator &it) noexcept
- iterator & operator= (const typename LinkedList< T >::const_iterator &it) noexcept
- iterator (typename LinkedList< T >::const_iterator &&it) noexcept
- iterator & operator= (typename LinkedList< T >::const_iterator &&it) noexcept
- iterator (const iterator &other) noexcept=default
- iterator & operator= (const iterator &other) noexcept=default
- iterator (iterator &&other) noexcept=default
- iterator & operator= (iterator &&other) noexcept=default
- LinkedListNode< T > * getNode () const noexcept

5.4.1 Detailed Description

template<typename T, typename Allocator = std::allocator<LinkedListNode<T>>>
class LinkedList< T, Allocator >::iterator

A bidirectional iterator for traversing elements in the linked list.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 iterator() [1/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::iterator::iterator ( )  [inline], [noexcept]
```

Default constructor.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.2.2 iterator() [2/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::iterator::iterator (
            LinkedListNode< T > * node )  [inline], [explicit], [noexcept]
```

Constructor.

Parameters

node	Pointer to the `LinkedListNode` to initialize the iterator with.

Postcondition

The iterator is constructed with the given `LinkedListNode` as the current node.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.2.3 iterator() [3/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::iterator::iterator (
            const typename LinkedList< T >::const_iterator & it )  [inline], [noexcept]
```

Copy constructibility from const_iterator.

Parameters

it	The constant iterator to copy construct from.

Postcondition

The iterator is constructed with the same current node as the constant iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.2.4 iterator() [4/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::iterator::iterator (
            typename LinkedList< T >::const_iterator && it )  [inline], [noexcept]
```

Move constructibility from const_iterator.

Parameters

it	The constant iterator to move construct from.

Postcondition

The iterator is constructed with the same current node as the constant iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.2.5 iterator() [5/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::iterator::iterator (
            const iterator & other )  [default], [noexcept]
```

Copy constructor.

Parameters

other	The iterator to be copied.

Postcondition

The iterator is constructed as a copy of the other iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.2.6 iterator() [6/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::iterator::iterator (
            iterator && other )  [default], [noexcept]
```

Move constructor.

Parameters

other	The iterator to be moved.

Postcondition

The iterator is constructed by moving the other iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.3 Member Function Documentation

5.4.3.1 getNode()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedListNode<T>* LinkedList< T, Allocator >::iterator::getNode ( ) const  [inline], [noexcept]
```

Get the Node where the iterator is pointing.

Returns

A pointer to the current node where the iterator is pointing.

Postcondition

Returns a pointer to the current node where the iterator is pointing.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.3.2 operator"!=()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
bool LinkedList< T, Allocator >::iterator::operator!= (
            const iterator & other ) const  [inline], [noexcept]
```

Inequality comparison operator for the iterator.

Parameters

other	The iterator to compare with.

Returns

True if both iterators do not point to the same node, otherwise false.

Postcondition

Checks if the current node of this iterator is not equal to the current node of the other iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.3.3 operator*()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
T& LinkedList< T, Allocator >::iterator::operator* ( ) const  [inline], [noexcept]
```

Dereference operator for the iterator.

Returns

A reference to the data of the current node.

Postcondition

Returns a reference to the data of the current node.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.3.4 operator++() [1/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator& LinkedList< T, Allocator >::iterator::operator++ ( )  [inline], [noexcept]
```

Pre-increment operator for the iterator.

Returns

A reference to the iterator after the increment.

Postcondition

Moves the iterator to the next node in the linked list.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.3.5 operator++() [2/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator LinkedList< T, Allocator >::iterator::operator++ (
            int  )  [inline], [noexcept]
```

Post-increment operator for the iterator.

Returns

An iterator pointing to the previous position before the increment (this).

Postcondition

Moves the iterator to the next node in the linked list.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.3.6 operator--() [1/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator& LinkedList< T, Allocator >::iterator::operator-- ( )  [inline]
```

Pre-decrement operator for the iterator.

Returns

A reference to the iterator after the decrement.

Postcondition

Moves the iterator to the previous node in the linked list.

5.4.3.7 operator--() [2/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator LinkedList< T, Allocator >::iterator::operator-- (
            int  )  [inline]
```

Post-decrement operator for the iterator.

Returns

An iterator pointing to the previous position before the decrement (this).

Postcondition

Moves the iterator to the previous node in the linked list.

5.4.3.8 operator->()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
T* LinkedList< T, Allocator >::iterator::operator-> ( ) const  [inline], [noexcept]
```

Arrow operator for the iterator.

Returns

A pointer to the data of the current node.

Postcondition

Returns a pointer to the data of the current node.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.3.9 operator=() [1/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator& LinkedList< T, Allocator >::iterator::operator= (
            const iterator & other )  [default], [noexcept]
```

Copy assignment operator.

Parameters

other	The iterator to be copied.

Returns

A reference to the iterator after the assignment.

Postcondition

The iterator is assigned as a copy of the other iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.3.10 operator=() [2/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator& LinkedList< T, Allocator >::iterator::operator= (
            const typename LinkedList< T >::const_iterator & it )  [inline], [noexcept]
```

Copy assignment from const_iterator.

Parameters

it	The constant iterator to copy assign from.

Returns

A reference to the iterator after the assignment.

Postcondition

The iterator is assigned with the same current node as the constant iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.3.11 operator=() [3/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator& LinkedList< T, Allocator >::iterator::operator= (
            iterator && other )  [default], [noexcept]
```

Move assignment operator.

Parameters

other	The iterator to be moved.

Returns

A reference to the iterator after the assignment.

Postcondition

The iterator is assigned by moving the other iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.3.12 operator=() [4/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator& LinkedList< T, Allocator >::iterator::operator= (
            typename LinkedList< T >::const_iterator && it )  [inline], [noexcept]
```

Move assignment from const_iterator.

Parameters

it	The constant iterator to move assign from.

Returns

A reference to the iterator after the assignment.

Postcondition

The iterator is assigned with the same current node as the constant iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.4.3.13 operator==()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
bool LinkedList< T, Allocator >::iterator::operator== (
            const iterator & other ) const  [inline], [noexcept]
```

Equality comparison operator for the iterator.

Parameters

other	The iterator to compare with.

Returns

True if both iterators point to the same node, otherwise false.

Postcondition

Checks if the current node of this iterator is equal to the current node of the other iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

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

- BagContainerAdaptor/include/BagContainerAdaptor/linked_list.hpp

5.5 LinkedList< T, Allocator > Class Template Reference

`#include <linked_list.hpp>`

Classes

- class const_iterator

  A bidirectional constant iterator for traversing elements in the linked list.
- class const_reverse_iterator

  A bidirectional constant reverse iterator for traversing items backwards in the linked list.
- class iterator

  A bidirectional iterator for traversing elements in the linked list.
- class reverse_iterator

  A bidirectional reverse iterator for traversing items backwards in the linked list.

Public Types

- using value_type = T

  The type of items stored in the linked list.

Public Member Functions

- LinkedList () noexcept
- ~LinkedList () noexcept
- LinkedList (std::initializer_list< value_type > list)
- LinkedList (LinkedList &&other) noexcept
- LinkedList & operator= (LinkedList &&other) noexcept
- LinkedList (const LinkedList< T > &other) noexcept
- LinkedList & operator= (const LinkedList< T > &other) noexcept
- iterator begin () noexcept
- iterator end () noexcept
- const_iterator cbegin () const noexcept
- const_iterator cend () const noexcept
- reverse_iterator rbegin () noexcept
- reverse_iterator rend () noexcept
- const_reverse_iterator crbegin () const noexcept
- const_reverse_iterator crend () const noexcept
- void clear () noexcept
- iterator insert (const T &value)
- iterator insert (iterator pos, const T &value)
- iterator erase (const T &value)
- iterator erase (iterator pos)
- iterator erase (iterator first, iterator last)
- void swap (LinkedList &other) noexcept
- iterator find (const T &value) noexcept
- iterator find (const T &value) const noexcept
- T & front () noexcept
- const T & front () const noexcept
- T & back () noexcept
- const T & back () const noexcept
- size_t size () const noexcept
- bool empty () const

5.5.1 Detailed Description

template<typename T, typename Allocator = std::allocator<LinkedListNode<T>>>
class LinkedList< T, Allocator >

This linked list is stl compatible a doubly linked list containing basic functionality for container and four different iterator types.

Template Parameters

T	The type of elements stored in the linked list.
Allocator	The type of allocator used in this linked list, initialized as std::allocator by default.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 LinkedList() [1/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::LinkedList ( )  [inline], [noexcept]
```

Default constructor.

Postcondition

Constructs a new `LinkedList` object with no elements.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.2.2 ~LinkedList()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::~LinkedList ( )  [inline], [noexcept]
```

Destructor.

Postcondition

Destroys the `LinkedList` object, freeing all associated resources.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.2.3 LinkedList() [2/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
```

```
LinkedList< T, Allocator >::LinkedList (
            std::initializer_list< value_type > list )  [inline]
```

Initializer list constructor.

Parameters

list	An initializer list containing values to initialize the LinkedList with.

Postcondition

Constructs a new LinkedList object with elements from the initializer list.

Exceptions

The	`insert` function may throw exceptions if memory allocation fails or if an exception is thrown by the element's constructor.

5.5.2.4 LinkedList() [3/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::LinkedList (
            LinkedList< T, Allocator > && other )  [inline], [noexcept]
```

Move constructor.

Parameters

other	The LinkedList to be moved from.

Postcondition

Constructs a new LinkedList by moving the content from the other LinkedList. The other LinkedList will be left in a valid but unspecified state.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.2.5 LinkedList() [4/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::LinkedList (
            const LinkedList< T > & other )  [inline], [noexcept]
```

Copy constructor.

Parameters

other	The LinkedList to be copied from.

Postcondition

Constructs a new LinkedList by copying the content from the other LinkedList.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3 Member Function Documentation

5.5.3.1 back() [1/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const T& LinkedList< T, Allocator >::back ( ) const  [inline], [noexcept]
```

Returns a constant reference to the last element in the linked list in const context.

Returns

A constant reference to the last element.

Precondition

The linked list must not be empty.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.2 back() [2/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
T& LinkedList< T, Allocator >::back ( )  [inline], [noexcept]
```

Returns a reference to the last element in the linked list.

Returns

A reference to the last element.

Precondition

The linked list must not be empty.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.3 begin()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator LinkedList< T, Allocator >::begin ( )  [inline], [noexcept]
```

Get an iterator to the beginning of the linked list.

Returns

An iterator pointing to the first element in the linked list.

Note

If the linked list is empty, the iterator will be equal to the end iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.4 cbegin()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_iterator LinkedList< T, Allocator >::cbegin ( ) const  [inline], [noexcept]
```

Get a const iterator to the beginning of the linked list.

Returns

A const iterator pointing to the first element in the linked list.

Note

If the linked list is empty, the iterator will be equal to the end iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.5 cend()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_iterator LinkedList< T, Allocator >::cend ( ) const  [inline], [noexcept]
```

Get a const iterator to the end of the linked list.

Returns

A const iterator pointing to the position past the last element in the linked list.

Note

This constant iterator acts as a sentinel and should not be dereferenced.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.6 clear()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
void LinkedList< T, Allocator >::clear ( )  [inline], [noexcept]
```

Clear the elements in the LinkedList and deallocate memory.

Postcondition

Removes all elements from the LinkedList, and deallocates the memory used by each element.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.7 crbegin()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
```

```
const_reverse_iterator LinkedList< T, Allocator >::crbegin ( ) const  [inline], [noexcept]
```

Get a constant reverse iterator to the beginning of the linked list.

Returns

A constant reverse iterator pointing past the first element in the linked list.

Note

If constant reverse iterator acts as a sentinel and should not be dereferenced.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.8 crend()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const_reverse_iterator LinkedList< T, Allocator >::crend ( ) const  [inline], [noexcept]
```

Get a constant reverse iterator to end of the linked list.

Returns

A constant reverse iterator pointing to the last element in the linked list.

Note

If linked list is empty, this constant reverse iterator will be equal to the crend iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.9 empty()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
bool LinkedList< T, Allocator >::empty ( ) const  [inline]
```

Checks whether the linked list is empty.

Returns

True if the linked list is empty, otherwise false.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.10 end()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator LinkedList< T, Allocator >::end ( )  [inline], [noexcept]
```

Get an iterator to the end of the linked list.

Returns

An iterator pointing to the position past the last element in the linked list.

Note

This iterator acts as a sentinel and should not be dereferenced.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.11 erase() [1/3]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator LinkedList< T, Allocator >::erase (
            const T & value )  [inline]
```

Remove all occurrences of the specified value from the linked list.

Parameters

value	The value of the elements to be removed.

Returns

An iterator that points to the element following the last removed element, or the end() iterator if no element was removed.

Precondition

The value_type of the linked list must support equality comparison.

Postcondition

All elements with the specified value are removed from the linked list.

Exceptions

May	throw an exception if memory deallocation fails (depends on the allocator).

5.5.3.12 erase() [2/3]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator LinkedList< T, Allocator >::erase (
            iterator first,
            iterator last ) [inline]
```

Remove the elements in the range [first, last] from the linked list.

Parameters

first	An iterator pointing to the first element of the range to be removed.
last	An iterator pointing to the element just beyond the last element of the range to be removed.

Returns

An iterator that points to the element following the last removed element, or the end() iterator if the last element was removed.

Precondition

The provided iterators must be valid and dereferenceable.

The range [first, last] must be a valid range within the linked list.

Postcondition

The elements in the range [first, last] are removed from the linked list.

The iterator following the last removed element is returned.

Exceptions

May	throw an exception if the deallocation of memory fails.

5.5.3.13 erase() [3/3]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
```

```
iterator LinkedList< T, Allocator >::erase (
            iterator pos )  [inline]
```

Remove the element at the specified position in the linked list.

Parameters

pos	An iterator pointing to the element to be removed.

Returns

An iterator that points to the element following the removed element, or the end() iterator if the last element was removed.

Precondition

The provided iterator must be valid and dereferenceable.

The linked list must not be empty.

Postcondition

The element at the position specified by the iterator is removed from the linked list.

The iterator following the removed element is returned.

Exceptions

May	throw an exception if memory deallocation fails (depends on the allocator).

5.5.3.14 find() [1/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator LinkedList< T, Allocator >::find (
            const T & value ) const  [inline], [noexcept]
```

Find the first occurrence of a value in the linked list in a constant context.

Parameters

value	The value to search for.

Returns

An iterator to the first occurrence of the value in the linked list, or the end() iterator if the value is not found.

Precondition

The linked list must not be empty.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.15 find() [2/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator LinkedList< T, Allocator >::find (
            const T & value )  [inline], [noexcept]
```

Find the first occurrence of a value in the linked list.

Parameters

value	The value to search for.

Returns

An iterator to the first occurrence of the value in the linked list, or the end() iterator if the value is not found.

Precondition

The linked list must not be empty.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.16 front() [1/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
const T& LinkedList< T, Allocator >::front ( ) const  [inline], [noexcept]
```

Returns a constant reference to the first element in the linked list in const context.

Returns

A constant reference to the first element.

Precondition

The linked list must not be empty.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.17 front() [2/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
T& LinkedList< T, Allocator >::front ( )  [inline], [noexcept]
```

Returns a reference to the first element in the linked list.

Returns

A reference to the first element.

Precondition

The linked list must not be empty.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.18 insert() [1/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator LinkedList< T, Allocator >::insert (
            const T & value )  [inline]
```

Insert a new element with the given value at the end of the linked list.

Parameters

value	The value of the element to be inserted.

Returns

An iterator that points to the newly inserted element.

Precondition

The value_type of the linked list must be copy constructible.

Postcondition

The element with the specified value is inserted at the end of the linked list.

Exceptions

May	throw std::bad_alloc if memory allocation fails during the operation.

Note

If the value_type of the linked list is not copy constructible, this function will not compile.

5.5.3.19 insert() [2/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
iterator LinkedList< T, Allocator >::insert (
            iterator pos,
            const T & value )  [inline]
```

Insert a new element with the given value at the specified position in the linked list.

Parameters

pos	An iterator pointing to the position where the element is inserted.
value	The value of the element to be inserted.

Returns

An iterator that points to the newly inserted element.

Precondition

The value_type of the linked list must be copy constructible.

The iterator `pos` must be a valid iterator within the linked list.

Postcondition

The element with the specified value is inserted at the position indicated by `pos`.

Exceptions

May	throw std::bad_alloc if memory allocation fails during the operation.

Note

If the value_type of the linked list is not copy constructible, this function will not compile.

5.5.3.20 operator=() [1/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList& LinkedList< T, Allocator >::operator= (
            const LinkedList< T > & other )  [inline], [noexcept]
```

Copy assignment operator.

Parameters

other	The LinkedList to be assigned from.

Returns

A reference to the LinkedList after the attempted copy assignment.

Postcondition

Copies the content from the other LinkedList to this LinkedList.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.21 operator=() [2/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList& LinkedList< T, Allocator >::operator= (
            LinkedList< T, Allocator > && other )  [inline], [noexcept]
```

Move assingment operator.

Parameters

other	The `LinkedList` to be moved from.

Returns

A reference to the LinkedList after the move assignment.

Postcondition

Moves the content from the other LinkedList to this LinkedList. The other LinkedList will be left in a valid but unspecified state.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.22 rbegin()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
reverse_iterator LinkedList< T, Allocator >::rbegin ( )  [inline], [noexcept]
```

Get a reverse iterator to the beginning of the linked list.

Returns

A reverse iterator pointing past the first element in the linked list.

Note

This reverse iterator acts as a sentinel and should not be dereferenced.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.23 rend()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
reverse_iterator LinkedList< T, Allocator >::rend ( )  [inline], [noexcept]
```

Get a reverse iterator to the beginning of the linked list.

Returns

A reverse iterator pointing to the last element in the linked list.

Note

If linked list is empty, this reverse iterator will be equal to the crbegin iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.24 size()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
size_t LinkedList< T, Allocator >::size ( ) const  [inline], [noexcept]
```

Returns the number of elements in the linked list.

Returns

The number of elements in the linked list.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.5.3.25 swap()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
void LinkedList< T, Allocator >::swap (
            LinkedList< T, Allocator > & other )  [inline], [noexcept]
```

Swap the contents of this linked list with another linked list.

Parameters

other	The other linked list to swap with.

Postcondition

The contents of this linked list are exchanged with the contents of the other linked list.

Exceptions

noexcept	No exceptions are thrown by this operation.

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

- BagContainerAdaptor/include/BagContainerAdaptor/linked_list.hpp

5.6 LinkedListNode< T > Struct Template Reference

```
#include <linked_list.hpp>
```

Public Member Functions

- LinkedListNode (const T &value) noexcept

Public Attributes

- T m_data

  The data inside the Node.
- LinkedListNode< T > * m_next = nullptr

  Pointing towards the next item in the linked list.
- LinkedListNode< T > * m_inverse = nullptr

  Pointing towards the previous item in the linked list.

5.6.1 Detailed Description

template<typename T>
struct LinkedListNode< T >

LinkedListNode represents a single node in the linked list.

Template Parameters

T	The type of data stored in the node.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 LinkedListNode()

```
template<typename T >
LinkedListNode< T >::LinkedListNode (
            const T & value )  [inline], [explicit], [noexcept]
```

Constructor.

Parameters

value	The value of the data in the Node.

Postcondition

The `m_data` is initialized with value.

Exceptions

noexcept	No exceptions are thrown by this operation.

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

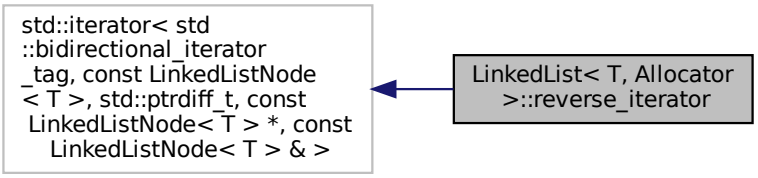
- BagContainerAdaptor/include/BagContainerAdaptor/linked_list.hpp

5.7 LinkedList< T, Allocator >::reverse_iterator Class Reference

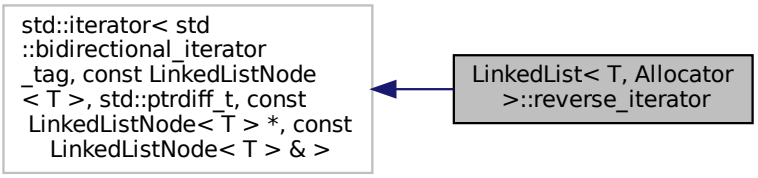
A bidirectional reverse iterator for traversing items backwards in the linked list.

`#include <linked_list.hpp>`

Inheritance diagram for LinkedList< T, Allocator >::reverse_iterator:



Collaboration diagram for LinkedList< T, Allocator >::reverse_iterator:



Public Member Functions

- reverse_iterator () noexcept
- reverse_iterator (LinkedListNode< T > *node) noexcept
- T & operator* () const noexcept
- T * operator-> () const noexcept
- reverse_iterator & operator++ ()

- reverse_iterator operator++ (int)
- reverse_iterator & operator-- ()
- reverse_iterator operator-- (int)
- bool operator== (const reverse_iterator &other) const noexcept
- bool operator!= (const reverse_iterator &other) const noexcept
- reverse_iterator (const typename LinkedList< T >::const_reverse_iterator &it) noexcept
- reverse_iterator & operator= (const typename LinkedList< T >::const_reverse_iterator &it) noexcept
- reverse_iterator (typename LinkedList< T >::const_reverse_iterator &&it) noexcept
- reverse_iterator & operator= (typename LinkedList< T >::const_reverse_iterator &&it) noexcept
- reverse_iterator (const reverse_iterator &other) noexcept=default
- reverse_iterator & operator= (const reverse_iterator &other) noexcept=default
- reverse_iterator (reverse_iterator &&other) noexcept=default
- reverse_iterator & operator= (reverse_iterator &&other) noexcept=default
- LinkedListNode< T > * getNode () const noexcept

5.7.1 Detailed Description

template<typename T, typename Allocator = std::allocator<LinkedListNode<T>>>
class LinkedList< T, Allocator >::reverse_iterator

A bidirectional reverse iterator for traversing items backwards in the linked list.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 reverse_iterator() [1/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::reverse_iterator::reverse_iterator ( )  [inline], [noexcept]
```

Default constructor.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.2.2 reverse_iterator() [2/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::reverse_iterator::reverse_iterator (
            LinkedListNode< T > * node )  [inline], [explicit], [noexcept]
```

Constructor.

Parameters

node	Pointer to the `LinkedListNode` to initialize reverse iterator with.

Postcondition

The reverse iterator is constructed with the given `LinkedListNode` as the current node.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.2.3 reverse_iterator() [3/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::reverse_iterator::reverse_iterator (
            const typename LinkedList< T >::const_reverse_iterator & it )  [inline], [noexcept]
```

Copy constructibility from constant reverse iterator.

Parameters

it	The constant reverse iterator to copy construct from.

Postcondition

The reverse iterator is constructed with the same current node as the constant reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.2.4 reverse_iterator() [4/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::reverse_iterator::reverse_iterator (
            typename LinkedList< T >::const_reverse_iterator && it )  [inline], [noexcept]
```

Move constructibility from constant reverse iterator.

Parameters

it	The constant reverse iterator to move construct from.

Postcondition

The reverse iterator is constructed with the same current node as the constant reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.2.5 reverse_iterator() [5/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::reverse_iterator::reverse_iterator (
            const reverse_iterator & other )  [default], [noexcept]
```

Copy constructor.

Parameters

other	The reverse iterator to be copied.

Postcondition

The reverse iterator is constructed as a copy of the other reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.2.6 reverse_iterator() [6/6]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedList< T, Allocator >::reverse_iterator::reverse_iterator (
            reverse_iterator && other )  [default], [noexcept]
```

Move constructor

Parameters

other	The reverse iterator to be moved.

Postcondition

The reverse iterator is constructed by moving the other reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.3 Member Function Documentation

5.7.3.1 getNode()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
LinkedListNode<T>* LinkedList< T, Allocator >::reverse_iterator::getNode ( ) const  [inline],
[noexcept]
```

Get the node where the iterator is pointing.

Returns

A pointer to the current node where the reverse iterator is pointing.

Postcondition

Returns a pointer to the current node where the reverse iterator is pointing.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.3.2 operator"!=()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
bool LinkedList< T, Allocator >::reverse_iterator::operator!= (
            const reverse_iterator & other ) const  [inline], [noexcept]
```

Inequality comparison operator for the iterator.

Parameters

other	The reverse iterator to compare with.

Returns

True if both reverse iterators do not point to the same node, otherwise false.

Postcondition

Checks if the current node of this reverse iterator is not equal to the current node of the other reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.3.3 operator∗()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
T& LinkedList< T, Allocator >::reverse_iterator::operator* ( ) const  [inline], [noexcept]
```

Dereference operator for the reverse iterator.

Returns

A reference to the data of the current node.

Postcondition

Returns a reference to the data of the current node.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.3.4 operator++() [1/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
reverse_iterator& LinkedList< T, Allocator >::reverse_iterator::operator++ ( )  [inline]
```

Pre-increment operator for the reverse iterator.

Returns

A reference to the reverse iterator after the increment.

Postcondition

Moves the reverse iterator to the previous node in the linked list.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.3.5 operator++() [2/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
reverse_iterator LinkedList< T, Allocator >::reverse_iterator::operator++ (
            int  )  [inline]
```

Post-increment operator for the reverse iterator.

Returns

A reference to the iterator after the increment.

Postcondition

Moves the iterator to the previous node in the linked list.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.3.6 operator--() [1/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
reverse_iterator& LinkedList< T, Allocator >::reverse_iterator::operator-- ( )  [inline]
```

Pre-decrement operator for the reverse iterator.

Returns

A reference to the reverse iterator after the decrement.

Postcondition

Moves the reverse iterator to the next node in the linked list.

5.7.3.7 operator--() [2/2]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
reverse_iterator LinkedList< T, Allocator >::reverse_iterator::operator-- (
            int  )  [inline]
```

Post decrement operator for the reverse linked list.

Returns

A reference to the reverse iterator after the decrement.

Postcondition

Moves the reverse iterator to the next node in the linked list.

5.7.3.8 operator->()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
T* LinkedList< T, Allocator >::reverse_iterator::operator-> ( ) const  [inline], [noexcept]
```

Arrow operator for the reverse iterator.

Returns

A pointer to the data of the current node.

Postcondition

Returns a pointer to the data of the current node.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.3.9 operator=() [1/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
reverse_iterator& LinkedList< T, Allocator >::reverse_iterator::operator= (
            const reverse_iterator & other )  [default], [noexcept]
```

Copy assignment operator.

Parameters

other	The reverse iterator to be copied.

Returns

A reference to the reverse iterator after the assignment.

Postcondition

The reverse iterator is assigned as a copy of the other reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.3.10 operator=() [2/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
reverse_iterator& LinkedList< T, Allocator >::reverse_iterator::operator= (
            const typename LinkedList< T >::const_reverse_iterator & it )  [inline], [noexcept]
```

Copy assignment from constant reverse iterator.

Parameters

it	The constant reverse iterator to copy assign from.

Returns

A reference to the reverse iterator after the assignment.

Postcondition

The reverse iterator is assigned with the same current node as the constant reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.3.11 operator=() [3/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
```

```
reverse_iterator& LinkedList< T, Allocator >::reverse_iterator::operator= (
            reverse_iterator && other )  [default], [noexcept]
```

Move assignment operator.

Parameters

other	The reverse iterator to be moved.

Returns

A reference to the reverse iterator after the assignment.

Postcondition

The reverse iterator is assigned by moving the other reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.3.12 operator=() [4/4]

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
reverse_iterator& LinkedList< T, Allocator >::reverse_iterator::operator= (
            typename LinkedList< T >::const_reverse_iterator && it )  [inline], [noexcept]
```

Move assignment from constant reverse iterator.

Parameters

it	The constant reverse iterator to move assign from.

Returns

A reference to the reverse iterator after the assignment.

Postcondition

The reverse iterator is assigned with the same current node as the constant reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

5.7.3.13 operator==()

```
template<typename T , typename Allocator = std::allocator<LinkedListNode<T>>>
bool LinkedList< T, Allocator >::reverse_iterator::operator== (
            const reverse_iterator & other ) const  [inline], [noexcept]
```

Equality comparison operator for the iterator.

Parameters

other	The reverse iterator to compared with.

Returns

True of both reverse iterators point to the same node, otherwise false.

Postcondition

Checks if the current node of this reverse iterator is equal to the current node of the other reverse iterator.

Exceptions

noexcept	No exceptions are thrown by this operation.

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

- BagContainerAdaptor/include/BagContainerAdaptor/linked_list.hpp