My Project

Generated by Doxygen 1.8.13

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 bst< K, V, CO >::_iterator< oK, oV > Class Template Reference

Implements the iterator for the bst.

```
#include <iterators.h>
```

Public Types

- using **value_type** = std::pair< oK, oV >
- using **reference** = value_type &
- using **pointer** = value_type ∗
- using **difference_type** = std::ptrdiff_t
- using **iterator_category** = std::forward_iterator_tag
- using **value_type** = std::pair< oK, oV >
- using **reference** = value_type &
- using **pointer** = value_type ∗
- using **difference_type** = std::ptrdiff_t
- using **iterator_category** = std::forward_iterator_tag

Public Member Functions

- _iterator ()=default

  Default constructor of iterator class.
- _iterator (node ∗p)

  Custom constructor of iterator class.
- ∼_iterator ()=default

  Default destructor of iterator class.
- _iterator & operator++ ()

  Pre-increment.
- _iterator operator++ (int)

  Post-increment.
- bool operator== (const _iterator &other_it) const

  Equality operator.
- bool operator!= (const _iterator &other_it) const

Inequality operator.
- reference operator∗ ()

  Dereference operator.
- pointer operator-> ()

  Arrow operator.
- _iterator ()=default

  Default constructor of the class _iterator.
- _iterator (node ∗p)

  Custom constructor of the class _iterator.
- **_iterator** (const _iterator &other_it)
- ∼_iterator ()=default

  Default destructor of the class _iterator.
- _iterator & operator++ ()

  Overload of the pre-increment operator ++.
- _iterator operator++ (int)

  Overload of the post-increment operator ++.
- bool operator== (const _iterator &other_it) const

  Overload of the operator ==.
- bool operator!= (const _iterator &other_it) const

  Overload of the operator !=.
- reference operator∗ ()

  Overload of the arrow operator ->
- pointer operator-> ()

  Overload of the dereference operator ∗.

Friends

- class **bst**
- std::ostream & operator<< (std::ostream &os, const _iterator &it)

  Overload of the put-to operator, which lets the user print the node pointed to by the iterator.
- std::ostream & **operator**<< (std::ostream &os, const _iterator &it)

3.1.1 Detailed Description

template<class K, class V, class CO = std::less<K>>
template<class oK, class oV>
class bst< K, V, CO >::_iterator< oK, oV >

Implements the iterator for the bst.

Template Parameters

oK	type of the key
oV	type of the value

3.1.2 Constructor & Destructor Documentation

3.1.2.1 _iterator() [1/2]

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
bst< K, V, CO >::_iterator< oK, oV >::_iterator (
             node * p )  [inline], [explicit]
```

Custom constructor of iterator class.

Parameters

p	pointer to a node

3.1.2.2 _iterator() [2/2]

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
bst< K, V, CO >::_iterator< oK, oV >::_iterator (
             node * p )  [inline], [explicit]
```

Custom constructor of the class _iterator.

Parameters

p	Raw pointer to node

Creates an iterator pointing to the given node

3.1.3 Member Function Documentation

3.1.3.1 operator"!=() [1/2]

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
bool bst< K, V, CO >::_iterator< oK, oV >::operator!= (
             const _iterator< oK, oV > & other_it ) const  [inline]
```

Overload of the operator !=.

Parameters

other↩_it	Iterator to be compared to

Returns

bool False if the iterators point to the same node, True otherwise

3.1.3.2 operator"!=() [2/2]

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
bool bst< K, V, CO >::_iterator< oK, oV >::operator!= (
             const _iterator< oK, oV > & other_it ) const  [inline]
```

Inequality operator.

Overloading of inequality operator

3.1.3.3 operator*() [1/2]

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
reference bst< K, V, CO >::_iterator< oK, oV >::operator* ( )  [inline]
```

Overload of the arrow operator ->

Returns

Pointer to the current node the iterator is pointing to

3.1.3.4 operator*() [2/2]

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
reference bst< K, V, CO >::_iterator< oK, oV >::operator* ( )  [inline]
```

Dereference operator.

Overloading of dereference operator

3.1.3.5 operator++() [1/4]

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
_iterator& bst< K, V, CO >::_iterator< oK, oV >::operator++ ( )  [inline]
```

Overload of the pre-increment operator ++.

Returns

_iterator& pointing to the next node

Used to traverse the tree from the leftmost to the rightmost node in ascending key order

3.1.3.6 operator++() [2/4]

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
_iterator bst< K, V, CO >::_iterator< oK, oV >::operator++ (
             int  )  [inline]
```

Overload of the post-increment operator ++.

Returns

_iterator& before advancing to the next node

Used to traverse the tree from the leftmost to the rightmost node in ascending key order

3.1.3.7 operator++() [3/4]

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
_iterator& bst< K, V, CO >::_iterator< oK, oV >::operator++ ( )  [inline]
```

Pre-increment.

Overloading of pre-increment operator

3.1.3.8 operator++() [4/4]

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
_iterator bst< K, V, CO >::_iterator< oK, oV >::operator++ (
             int  )  [inline]
```

Post-increment.

Overloading of post-increment operator

3.1.3.9 operator->() [1/2]

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
pointer bst< K, V, CO >::_iterator< oK, oV >::operator-> ( )  [inline]
```

Overload of the dereference operator $*$.

Returns

Reference to the data of the node the iterator is pointing to

3.1.3.10 operator->() [2/2]

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
pointer bst< K, V, CO >::_iterator< oK, oV >::operator-> ( )  [inline]
```

Arrow operator.

Overloading of arrow operator

3.1.3.11 operator==() [1/2]

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
bool bst< K, V, CO >::_iterator< oK, oV >::operator== (
             const _iterator< oK, oV > & other_it ) const  [inline]
```

Overload of the operator ==.

Parameters

other_it	Iterator to be compared to

Returns

bool True if the iterators point to the same node, False otherwise

3.1.3.12 operator==() [2/2]

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
bool bst< K, V, CO >::_iterator< oK, oV >::operator== (
             const _iterator< oK, oV > & other_it ) const  [inline]
```

Equality operator.

Overloading of equality operator

3.1.4 Friends And Related Function Documentation

3.1.4.1 operator<<

```
template<class K, class V, class CO = std::less<K>>
template<class oK , class oV >
std::ostream& operator<< (
             std::ostream & os,
             const _iterator< oK, oV > & it )  [friend]
```

Overload of the put-to operator, which lets the user print the node pointed to by the iterator.

Parameters

os	Stream where to print the content of the node poited to by the iterator
it	iterator pointing to the node of interest

Returns

os Stream where the content has been sent

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

- include/bst.h
- Trash/iterators.h

3.2 bst< K, V, CO > Class Template Reference

Implementation of the type: binary search tree.

```
#include <bst.h>
```

Classes

- class _iterator

  Implements the iterator for the bst.
- struct node

  Definition of the node struct.

Public Types

- using iterator = _iterator< K, V >

  bst iterator is defined in the header iterator.h
- using **const_iterator** = _iterator< const K, const V >

Public Member Functions

- bst ()=default

  Default constructor of bst class.
- bst (const pair_type &pair)

  Custom constructor of bst class.
- bst (key_type &&_key, value_type &&_value)

  Custom constructor of bst class.
- ∼bst ()=default

  Default destructor of bst class.
- bst (const bst &to_copy)

  Copy constructor of bst class.
- bst & operator= (const bst &to_copy)

Copy assignment of bst class.
- bst (bst &&move_from)

    Move constructor of bst class.
- bst & operator= (bst &&move_from)

    Move assignment of bst class.
- std::pair< iterator, bool > insert (const pair_type &x)

    Inserts a new element in the tree.
- std::pair< iterator, bool > insert (pair_type &&x)

    Inserts a new element in the tree.
- template<class... Types>
  std::pair< iterator, bool > emplace (Types &&... args)

    Inserts a new element in the tree constructed in-place.
- void clear ()

    Clears the content of the tree without any memory leak.
- iterator begin () noexcept

    Points to the "smallest" node.
- const_iterator begin () const noexcept

    Points to the "smallest" node.
- const_iterator cbegin () const noexcept

    Points to the "smallest" node.
- iterator end () noexcept

    Points to one past the "biggest" node.
- const_iterator end () const noexcept

    Points to one past the "biggest" node.
- const_iterator cend () const noexcept

    Points to one past the "biggest" node.
- iterator find (const key_type &x)

    Finds a node in the bst given a key.
- const_iterator find (const key_type &x) const

    Finds a node in the bst given a key.
- reference operator[ ] (const key_type &x)

    Overload of the [] operator, which lets the user find a value given the key.
- reference operator[ ] (key_type &&x)

    Overload of the [] operator, which lets the user find a value given the key.
- void balance ()

    Balances the bst, i.e. re-structures the tree in order to minimize its depth.
- void erase_node (node ∗N)

    Erase a node from the tree.
- void erase (const key_type &key)

    Erase a node with a given key from the tree.
- unsigned int node_depth (const key_type &k)

    Makes depth of the node available for the user.
- unsigned int max_depth ()

    Returns maximum depth of the binary search tree.
- void print_2D ()

    Prints tree structure.
- template<class... Types>
  std::pair< typename bst< K, V, CO >::iterator, bool > **emplace** (Types &&... args)

Public Attributes

- CO comp

  Comparison operator.

Friends

- std::ostream & operator<< (std::ostream &os, const bst &x)

  Overload of the put-to operator, which lets the user print the tree in ascending order of the keys.
- auto get_root (const bst &x)

  Returns the key and value of the root of the given bst.

3.2.1 Detailed Description

template<class K, class V, class CO = std::less<K>>
class bst< K, V, CO >

Implementation of the type: binary search tree.

Template Parameters

K	Key type
V	Value type
CO	Comparison operator typer (default is std::less<K>)

3.2.2 Member Typedef Documentation

3.2.2.1 iterator

```
template<class K, class V, class CO = std::less<K>>
using bst< K, V, CO >::iterator = _iterator<K,V>
```

bst iterator is defined in the header iterator.h

Template class holding the bst iterator type

3.2.3 Constructor & Destructor Documentation

3.2.3.1 bst() [1/4]

```
template<class K, class V, class CO = std::less<K>>
bst< K, V, CO >::bst (
             const pair_type & pair )  [inline], [explicit]
```

Custom constructor of bst class.

Parameters

pair	contains the key and value of the first node with which to build the tree (root)

3.2.3.2 bst() [2/4]

```
template<class K, class V, class CO = std::less<K>>
bst< K, V, CO >::bst (
             key_type && _key,
             value_type && _value )  [inline]
```

Custom constructor of bst class.

Parameters

_key	contains the key of the first node with which to build the tree (root)
_value	contains the value of the first node with which to build the tree (root)

3.2.3.3 bst() [3/4]

```
template<class K, class V, class CO = std::less<K>>
bst< K, V, CO >::bst (
             const bst< K, V, CO > & to_copy )  [inline]
```

Copy constructor of bst class.

Parameters

to_copy	bst object to be copied

3.2.3.4 bst() [4/4]

```
template<class K, class V, class CO = std::less<K>>
bst< K, V, CO >::bst (
             bst< K, V, CO > && move_from )  [inline]
```

Move constructor of bst class.

Parameters

move_from	bst object to be moved

Creates a bst by moving the content of the input bst

3.2.4 Member Function Documentation

3.2.4.1 begin() [1/2]

```
template<class K , class V , class CO >
bst< K, V, CO >::iterator bst< K, V, CO >::begin ( ) [noexcept]
```

Points to the "smallest" node.

Returns

iterator an iterator pointing to the leftmost node, i.e. the one with the smallest key

3.2.4.2 begin() [2/2]

```
template<class K , class V , class CO >
bst< K, V, CO >::const_iterator bst< K, V, CO >::begin ( ) const [noexcept]
```

Points to the "smallest" node.

Returns

const_iterator a const_iterator pointing to the leftmost node i.e. the one with the smallest key

3.2.4.3 cbegin()

```
template<class K , class V , class CO >
bst< K, V, CO >::const_iterator bst< K, V, CO >::cbegin ( ) const [noexcept]
```

Points to the "smallest" node.

Returns

const_iterator a const_iterator pointing to the leftmost node, i.e. the one with the smallest key

3.2.4.4 cend()

```
template<class K , class V , class CO >
bst< K, V, CO >::const_iterator bst< K, V, CO >::cend ( ) const  [noexcept]
```

Points to one past the "biggest" node.

Returns

const_iterator a const_iterator pointing to one past the last node, i.e. the node past the one with the larger key

3.2.4.5 emplace()

```
template<class K, class V, class CO = std::less<K>>
template<class...  Types>
std::pair<iterator,bool> bst< K, V, CO >::emplace (
             Types &&...  args )
```

Inserts a new element in the tree constructed in-place.

Template Parameters

Types	types of the parameters passed to build a new element

Parameters

args	parameters passed to build a new element

Returns

std::pair<iterator,bool> the function returns a pair of: an iterator pointing to the inserted node; a bool which is true if a new node has been allocated, false if the node is already in the tree

3.2.4.6 end() [1/2]

```
template<class K , class V , class CO >
bst< K, V, CO >::iterator bst< K, V, CO >::end ( )  [noexcept]
```

Points to one past the "biggest" node.

Returns

iterator an iterator pointing to one past the last node, i.e. the node past the one with the larger key

3.2.4.7 end() [2/2]

```
template<class K , class V , class CO >
bst< K, V, CO >::const_iterator bst< K, V, CO >::end ( ) const  [noexcept]
```

Points to one past the "biggest" node.

Returns

const_iterator a const_iterator pointing to one past the last node, i.e. the node past the one with the larger key

3.2.4.8 erase()

```
template<class K , class V , class CO >
void bst< K, V, CO >::erase (
             const key_type & key )
```

Erase a node with a given key from the tree.

Parameters

key	Key of the none to be erased

3.2.4.9 erase_node()

```
template<class K , class V , class CO >
void bst< K, V, CO >::erase_node (
             bst< K, V, CO >::node * N )
```

Erase a node from the tree.

Parameters

N	pointer to a node of the tree

3.2.4.10 find() [1/2]

```
template<class K , class V , class CO >
bst< K, V, CO >::iterator bst< K, V, CO >::find (
             const key_type & x )
```

Finds a node in the bst given a key.

Parameters

x	Key to be found

Returns

iterator pointing to the node with that key if the key exists, otherwise it returns an iterator pointing to "nullptr"

3.2.4.11 find() [2/2]

```
template<class K , class V , class CO >
bst< K, V, CO >::const_iterator bst< K, V, CO >::find (
             const key_type & x ) const
```

Finds a node in the bst given a key.

Parameters

x	Key to be found

Returns

const_iterator pointing to the node with that key if the key exists, otherwise it returns a const_iterator pointing to "nullptr"

3.2.4.12 insert() [1/2]

```
template<class K , class V , class CO >
std::pair< typename bst< K, V, CO >::iterator, bool > bst< K, V, CO >::insert (
             const pair_type & x )
```

Inserts a new element in the tree.

Parameters

x	pair to be inserted of type std::pair<key, value>

Returns

std::pair<iterator,bool> The function returns a pair of: an iterator pointing to the inserted node, a bool which is true if a new node has been allocated, false if the key is already present in the tree

3.2.4.13 insert() [2/2]

```
template<class K , class V , class CO >
std::pair< typename bst< K, V, CO >::iterator, bool > bst< K, V, CO >::insert (
            pair_type && x )
```

Inserts a new element in the tree.

Parameters

x	pair to be inserted of type std::pair<key, value>

Returns

std::pair<iterator,bool> The function returns a pair of: an iterator pointing to the inserted node; a bool which is true if a new node has been allocated, false if the key is already present in the tree

3.2.4.14 max_depth()

```
template<class K , class V , class CO >
unsigned int bst< K, V, CO >::max_depth ( )
```

Returns maximum depth of the binary search tree.

Returns

unsigned int storing the depth of the deepest node of the tree

3.2.4.15 node_depth()

```
template<class K , class V , class CO >
unsigned int bst< K, V, CO >::node_depth (
            const key_type & k )
```

Makes depth of the node available for the user.

Parameters

k	key value associated with the node whose depth is requested

Returns

unsigned int storing the depth of the node identified by the key

3.2.4.16 operator=() [1/2]

```
template<class K , class V , class CO >
bst< K, V, CO > & bst< K, V, CO >::operator= (
             const bst< K, V, CO > & to_copy )
```

Copy assignment of bst class.

Parameters

to_copy	bst object to be copied

Returns

bst& reference to copied binary search tree

3.2.4.17 operator=() [2/2]

```
template<class K , class V , class CO >
bst< K, V, CO > & bst< K, V, CO >::operator= (
             bst< K, V, CO > && move_from )
```

Move assignment of bst class.

Parameters

move_from	bst object to be moved

Returns

bst& reference to moved binary search tree

3.2.4.18 operator[]() [1/2]

```
template<class K , class V , class CO >
bst< K, V, CO >::reference bst< K, V, CO >::operator[] (
             const key_type & x )
```

Overload of the [] operator, which lets the user find a value given the key.

Parameters

x	Key to be found

Returns

value_type& If the key exists returns a reference to the associated value. Otherwise it adds a new node containing the input key and the default value and returns a reference to the value

3.2.4.19 operator[]() [2/2]

```
template<class K , class V , class CO >
bst< K, V, CO >::reference bst< K, V, CO >::operator[] (
             key_type && x )
```

Overload of the [] operator, which lets the user find a value given the key.

Parameters

x	Key to be found

Returns

value_type& If the key exists returns a reference to the associated value. Otherwise it adds a new node containing the input key and the default value and returns a reference to the value

3.2.5 Friends And Related Function Documentation

3.2.5.1 get_root

```
template<class K, class V, class CO = std::less<K>>
auto get_root (
             const bst< K, V, CO > & x )  [friend]
```

Returns the key and value of the root of the given bst.

Parameters

x	bst to return the info

Returns

std::pair<key_type, value_type> pair containing the key and the value of the root node

3.2.5.2 operator<<

```
template<class K, class V, class CO = std::less<K>>
std::ostream& operator<< (
             std::ostream & os,
             const bst< K, V, CO > & x )  [friend]
```

Overload of the put-to operator, which lets the user print the tree in ascending order of the keys.

Parameters

os	Stream where to print the content of the tree
x	bst to be printed

Returns

os Stream where the content has been sent

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

- include/bst.h
- include/methods.h

3.3 bst< K, V, CO >::node Struct Reference

Definition of the node struct.

```
#include <node.h>
```

Public Member Functions

- node ()=default

  Default constructor of node struct.
- node (const pair_type &n)

  Custom constructor of node struct.
- node (const pair_type &n, node ∗new_parent)

  Custom destructor of node struct.
- ∼node () noexcept=default

  Default destructor of node struct.
- node ∗ findLowest () noexcept

  Finds the "smaller" element of the bst object that has the current node as root.
- node ∗ findUpper ()

  Finds the first right anchestor of the current node.
- unsigned int depth (unsigned int &&Depth=1)

  Compute the depth of the current node in the tree.
- node ()=default

  Default constructor for the class node.
- node (pair_type &n)

  Custom constructor for the class node.
- node (const pair_type &n, node ∗new_parent)

  Custom constructor for the class node.
- ∼node () noexcept=default

Default destructor of the class node.

- node (const std::unique_ptr< node > &copy_from)

    DA COMMENTARE.
- node ∗ findLowest () noexcept

    Finds the node with the lowest key in the tree.
- node ∗ findUpper ()

    DA COMMENTARE.
- node ∗ rightmost ()

    DA COMMENTARE.

Public Attributes

- pair_type data

    pair of a key and a value, stored in the element of the bst object
- std::unique_ptr< node > left

    Unique pointer to the left child of the current element.
- std::unique_ptr< node > right

    Unique pointer to the right child of the current element.
- node ∗ parent

    Pointer to the parent node of the current element.

3.3.1 Detailed Description

template<class K, class V, class CO = std::less<K>>
struct bst< K, V, CO >::node

Definition of the node struct.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 node() [1/5]

```
template<class K, class V, class CO = std::less<K>>
bst< K, V, CO >::node::node (
            const pair_type & n )  [inline], [explicit]
```

Custom constructor of node struct.

Parameters

n	pair of a key and a value to store in the element node

3.3.2.2 node() [2/5]

```
template<class K, class V, class CO = std::less<K>>
bst< K, V, CO >::node::node (
            const pair_type & n,
            node * new_parent )  [inline]
```

Custom destructor of node struct.

Parameters

n	pair of a key and a value to store in the element node
new_parent	pointer to a node, which will become the parent of the new element node

3.3.2.3 node() [3/5]

```
template<class K, class V, class CO = std::less<K>>
bst< K, V, CO >::node::node (
            pair_type & n )  [inline], [explicit]
```

Custom constructor for the class node.

Parameters

n	Data to be inserted in the node

Initializes a node with its data

3.3.2.4 node() [4/5]

```
template<class K, class V, class CO = std::less<K>>
bst< K, V, CO >::node::node (
            const pair_type & n,
            node * new_parent )  [inline]
```

Custom constructor for the class node.

Parameters

n	Data to be inserted in the node
new_parent	Parent of the node

Initializes a node with data and parent node

3.3.2.5 node() [5/5]

```
template<class K, class V, class CO = std::less<K>>
```

```
bst< K, V, CO >::node::node (
             const std::unique_ptr< node > & copy_from )  [inline], [explicit]
```

DA COMMENTARE.

Parameters

copy_from	

3.3.3 Member Function Documentation

3.3.3.1 depth()

```
template<class K, class V, class CO = std::less<K>>
unsigned int bst< K, V, CO >::node::depth (
             unsigned int && Depth = 1 )  [inline]
```

Compute the depth of the current node in the tree.

Returns

Depth unsigned int storing the depth of the current node in the tree

3.3.3.2 findLowest() [1/2]

```
template<class K, class V, class CO = std::less<K>>
node* bst< K, V, CO >::node::findLowest ( )  [inline], [noexcept]
```

Finds the node with the lowest key in the tree.

Returns

Raw pointer to the node with the smallest key

3.3.3.3 findLowest() [2/2]

```
template<class K, class V, class CO = std::less<K>>
node* bst< K, V, CO >::node::findLowest ( )  [inline], [noexcept]
```

Finds the "smaller" element of the bst object that has the current node as root.

Returns

node∗ pointer to the leftmost node of the bst object that has the current node as root

3.3.3.4 findUpper() [1/2]

```
template<class K, class V, class CO = std::less<K>>
node∗ bst< K, V, CO >::node::findUpper ( )  [inline]
```

DA COMMENTARE.

Returns

3.3.3.5 findUpper() [2/2]

```
template<class K, class V, class CO = std::less<K>>
node∗ bst< K, V, CO >::node::findUpper ( )  [inline]
```

Finds the first right anchestor of the current node.

Returns

node∗ pointer to the first anchestor node which stands on the left of the current node

3.3.3.6 rightmost()

```
template<class K, class V, class CO = std::less<K>>
node∗ bst< K, V, CO >::node::rightmost ( )  [inline]
```

DA COMMENTARE.

Returns

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

- include/bst.h
- Trash/node.h

Chapter 4

File Documentation

4.1 Trash/iterators.h File Reference

header containing the implementation of the bst iterator_ class

```
#include <memory>
#include <utility>
#include "bst.h"
```

Classes

- class bst< K, V, CO >::_iterator< oK, oV >

  Implements the iterator for the bst.

4.1.1 Detailed Description

header containing the implementation of the bst iterator_ class

Authors

Marco Sicklinger, Marco Sciorilli, Lorenzo Cavuoti

4.2 Trash/node.h File Reference

header containing the implementation of the binary search tree

```
#include <memory>
#include <utility>
#include "bst.h"
```

Classes

- struct bst< K, V, CO >::node

    Definition of the node struct.

4.2.1 Detailed Description

header containing the implementation of the binary search tree

header containing the implementation of the bst node struct

Author

Authors

Marco Sicklinger, Marco Sciorilli, Lorenzo Cavuoti

Index