Advanced Programming project - binary search tree

Generated by Doxygen 1.8.17

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 _iterator< node_t, T > Class Template Reference

```
#include <iterator.hpp>
```

Public Types

- using **v_type** = T
- using **reference** = v_type &
- using **pointer** = v_type ∗
- using **difference_type** = std::ptrdiff_t
- using **iterator_category** = std::forward_iterator_tag

Public Member Functions

- _iterator (node_t ∗p)

  Constructor of new _iterator object.
- ∼_iterator () noexcept=default

  Default deconstructor of _iterator object.
- reference operator∗ () const

  Dereference operator.
- pointer operator-> () const

  Reference operator.
- _iterator & operator++ ()

  Overloading of pre-increment operator ++.

Private Attributes

- node_t ∗ current

  Raw pointer to the current node of type node_t.

Friends

- bool operator== (_iterator &a, _iterator &b)

  Overloading of equality operator.
- bool operator!= (_iterator &a, _iterator &b)

  Overloading of inequality operator.

3.1.1 Detailed Description

template<typename node_t, typename T>
class _iterator< node_t, T >

Template Parameters

node←_t	template for an object of type node_t.
T	template for an object of type T that is the pair_type.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 _iterator()

```
template<typename node_t , typename T >
_iterator< node_t, T >::_iterator (
              node_t * p )  [inline], [explicit]
```

Constructor of new _iterator object.

Template Parameters

p	raw pointer to a node.

3.1.3 Member Function Documentation

3.1.3.1 operator*()

```
template<typename node_t , typename T >
reference _iterator< node_t, T >::operator* ( ) const  [inline]
```

Dereference operator.

Returns

Reference to the value stored by the node pointed by the iterator.

3.1.3.2 operator++()

```
template<typename node_t , typename T >
_iterator& _iterator< node_t, T >::operator++ ( )  [inline]
```

Overloading of pre-increment operator ++.

Allow for the traverse of the bst from begin() to end() so, from left to right.

Returns

Reference to the iterator.

3.1.3.3 operator->()

```
template<typename node_t , typename T >
pointer _iterator< node_t, T >::operator-> ( ) const  [inline]
```

Reference operator.

Returns

Pointer to the value stored by the node pointed by the iterator.

3.1.4 Friends And Related Function Documentation

3.1.4.1 operator"!=

```
template<typename node_t , typename T >
bool operator!= (
            _iterator< node_t, T > & a,
            _iterator< node_t, T > & b )  [friend]
```

Overloading of inequality operator.

Template Parameters

a	reference to the first iterator.
b	reference to the second iterator.

Returns

Bool true if they point to different nodes, false otherwise.

3.1.4.2 operator==

```
template<typename node_t , typename T >
bool operator== (
            _iterator< node_t, T > & a,
            _iterator< node_t, T > & b )  [friend]
```

Overloading of equality operator.

Template Parameters

a	reference to the first iterator.
b	reference to the second iterator.

Returns

Bool true if they point to the same node, false otherwise.

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

- iterator.hpp

3.2 bst< value_type, key_type, cmp_op > Class Template Reference

Public Member Functions

- bst () noexcept=default

  Default constructor for the bst class.
- ~bst () noexcept=default

  Default deconstructor for the bst class.
- std::pair< iterator, bool > insert (const pair_type &x)

  This function calls the _insert function to insert a new node in the tree, if not already present.
- std::pair< iterator, bool > insert (pair_type &&x)

  This function calls the _insert function using std::move() to insert a new node in the tree, if not already present.
- template<class... Types>
  std::pair< iterator, bool > emplace (Types &&... args)

  This function calls the insert function to insert a new node in the tree, if not already present, by both giving as input std::pair<key, value> and giving the key and the value.
- void clear () noexcept

  This functions clears the content of the tree by setting the root node to nullptr.
- iterator begin () noexcept

  This function searches for the left-most node of the tree and returns an iterator pointing to that node.
- const_iterator begin () const noexcept

  This function searches for the left-most node of the tree and returns a const iterator pointing to that node.
- const_iterator cbegin () const noexcept

  This function searches for the left-most node of the tree and returns a const iterator pointing to that node.
- iterator end () noexcept

  This function returns an iterator pointing to one past the last element of the tree, namely the right-most node.
- const_iterator end () const noexcept

This function returns a const iterator pointing to one past the last element of the tree, namely the right-most node.

- const_iterator cend () const noexcept

  This function returns a const iterator pointing to one past the last element of the tree, namely the right-most node.
- iterator find (const key_type &x)

  This function calls _find to check if a node is present in the tree by checking its key.
- const_iterator find (const key_type &x) const

  This function calls _find to check if a node is present in the tree by checking its key.
- bst (const bst &x)

  Copy constructor for bst that creates a deep copy of the tree by calling the node copy constructor on the root node.
- bst & operator= (const bst &x)

  Copy assignment for a bst tree.
- bst (bst &&x) noexcept=default

  Default move constructor for bst.
- bst & operator= (bst &&x) noexcept=default

  Default move assignment for bst.
- void balance ()

  This function is used to balance the tree, by means of the _balance function.
- value_type & operator[ ] (const key_type &x)

  Overloaded operator that search the key to return corresponding associated value.
- value_type & operator[ ] (key_type &&x)

  Overloaded operator that search the key to return corresponding associated value.
- void erase (const key_type &x)

  This function erases the content of the node with key equal to the input one, if present.

Private Types

- using **pair_type** = std::pair< const key_type, value_type >
- using **node_t** = node< pair_type >
- using **iterator** = _iterator< node_t, pair_type >
- using **const_iterator** = _iterator< node_t, const pair_type >

Private Member Functions

- node_t * _find (const key_type &x)

  This function checks if a node is present in the tree by checking its key.
- template<typename O >
  std::pair< iterator, bool > _insert (O &&x)

  This function inserts a new node in the tree, if not present.
- node_t * _inorder (node_t *x)

  This function searches for the left-most node in the tree rooted at the input node x.
- void _balance (std::vector< pair_type > &nodes, int start, int end)

  This functions calls recursively itself in order to balance the tree.

Private Attributes

- std::unique_ptr< node_t > root

  Definition of the unique pointer to the root node of the tree.
- cmp_op op

  Comparison operator.

Friends

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

  Friend operator that prints the tree ordered by key values using const iterators.

3.2.1 Constructor & Destructor Documentation

3.2.1.1 bst() [1/2]

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
bst< value_type, key_type, cmp_op >::bst (
            const bst< value_type, key_type, cmp_op > & x )  [inline]
```

Copy constructor for bst that creates a deep copy of the tree by calling the node copy constructor on the root node.

Template Parameters

x	const lvalue reference to the tree.

3.2.1.2 bst() [2/2]

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
bst< value_type, key_type, cmp_op >::bst (
            bst< value_type, key_type, cmp_op > && x )  [default], [noexcept]
```

Default move constructor for bst.

Template Parameters

x	rvalue reference to the tree.

3.2.2 Member Function Documentation

3.2.2.1 _balance()

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
void bst< value_type, key_type, cmp_op >::_balance (
            std::vector< pair_type > & nodes,
            int start,
            int end )  [inline], [private]
```

This functions calls recursively itself in order to balance the tree.

Template Parameters

nodes	reference to the vector containing the node pairs ordered by keys.
start	integer index of the first element of the vector.
end	integer index of the last element of the vector, namely its length.

3.2.2.2 _find()

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
node_t* bst< value_type, key_type, cmp_op >::_find (
             const key_type & x )  [inline], [private]
```

This function checks if a node is present in the tree by checking its key.

It iteratively compare the key of each node to the one of its right node, if it is bigger, or of its left node, if it is smaller.

Template Parameters

x	const lvalue reference to the key to look for.

Returns

Raw pointer to the found node or a nullptr if the key was not found.

3.2.2.3 _inorder()

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
node_t* bst< value_type, key_type, cmp_op >::_inorder (
             node_t * x )  [inline], [private]
```

This function searches for the left-most node in the tree rooted at the input node x.

If the input node does not have any left nodes, it simply returns the input node.

Template Parameters

x	raw pointer to the input node.

Returns

Raw pointer to the left-most node in the tree rooted at x.

3.2.2.4 _insert()

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
template<typename O >
std::pair<iterator, bool> bst< value_type, key_type, cmp_op >::_insert (
            O && x )  [inline], [private]
```

This function inserts a new node in the tree, if not present.

It first checks if the key is already present, by means of the _find function, and if the pair is not present in the tree, it searches for the right place to insert it.

Template Parameters

x	reference to the pair to be inserted.

Returns

std::pair<iterator,bool> iterator to the inserted node, true, or iterator to the already existing node, false.

3.2.2.5 balance()

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
void bst< value_type, key_type, cmp_op >::balance ( )  [inline]
```

This function is used to balance the tree, by means of the _balance function.

It first creates a vector containing the nodes pairs ordered by key values, clear the unbalanced tree and then calls _balance on the nodes vector to generate a balanced tree.

3.2.2.6 begin() [1/2]

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
const_iterator bst< value_type, key_type, cmp_op >::begin ( ) const  [inline], [noexcept]
```

This function searches for the left-most node of the tree and returns a const iterator pointing to that node.

Returns

const_iterator to the left-most node.

3.2.2.7 begin() [2/2]

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
iterator bst< value_type, key_type, cmp_op >::begin ( )  [inline], [noexcept]
```

This function searches for the left-most node of the tree and returns an iterator pointing to that node.

Returns

iterator to the left-most node.

3.2.2.8 cbegin()

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
const_iterator bst< value_type, key_type, cmp_op >::cbegin ( ) const  [inline], [noexcept]
```

This function searches for the left-most node of the tree and returns a const iterator pointing to that node.

Returns

Const iterator to the left-most node.

3.2.2.9 cend()

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
const_iterator bst< value_type, key_type, cmp_op >::cend ( ) const  [inline], [noexcept]
```

This function returns a const iterator pointing to one past the last element of the tree, namely the right-most node.

Returns

Const iterator to one past the last node.

3.2.2.10 emplace()

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
template<class...  Types>
std::pair<iterator,bool> bst< value_type, key_type, cmp_op >::emplace (
             Types &&...  args )  [inline]
```

This function calls the insert function to insert a new node in the tree, if not already present, by both giving as input std::pair<key, value> and giving the key and the value.

Template Parameters

args	a std::pair<key, value> or a key and value.

Returns

std::pair<iterator, bool> returned by the insert function.

3.2.2.11 end() [1/2]

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
const_iterator bst< value_type, key_type, cmp_op >::end ( ) const  [inline], [noexcept]
```

This function returns a const iterator pointing to one past the last element of the tree, namely the right-most node.

Returns

Const iterator to one past the last node.

3.2.2.12 end() [2/2]

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
iterator bst< value_type, key_type, cmp_op >::end ( )  [inline], [noexcept]
```

This function returns an iterator pointing to one past the last element of the tree, namely the right-most node.

Returns

Iterator to one past the last node.

3.2.2.13 erase()

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
void bst< value_type, key_type, cmp_op >::erase (
             const key_type & x )  [inline]
```

This function erases the content of the node with key equal to the input one, if present.

First of all, the function calls the _find function to check if the key is present or not in the tree. If not, the function returns. Then we might face three different situations, the node to be erased has no left and right nodes, namely it is a leaf, it might have only one left or right node, or it has both left and right nodes. In all the situations, we need to check if the node to be erased is the root node and release the node parent's ownership, and in the latter two cases, we also need to reset the ownership of the node to left and/or right nodes. When the node to be erased has two children nodes, the function will call _inorder to find the first inorder successor, and then calls itself to erase the successor node, after having stored its value.

Template Parameters

x	const lvalue of the key to look for.

3.2.2.14 find() [1/2]

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
iterator bst< value_type, key_type, cmp_op >::find (
            const key_type & x )  [inline]
```

This function calls _find to check if a node is present in the tree by checking its key.

Template Parameters

x	const lvalue reference to the key to look for.

Returns

Iterator pointing to the found node or to a null pointer if the key was not found.

3.2.2.15 find() [2/2]

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
const_iterator bst< value_type, key_type, cmp_op >::find (
            const key_type & x ) const  [inline]
```

This function calls _find to check if a node is present in the tree by checking its key.

Template Parameters

x	const lvalue reference to the key to look for.

Returns

Const iterator pointing to the found node or to a null pointer if the key was not found.

3.2.2.16 insert() [1/2]

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
std::pair<iterator, bool> bst< value_type, key_type, cmp_op >::insert (
            const pair_type & x )  [inline]
```

This function calls the _insert function to insert a new node in the tree, if not already present.

Template Parameters

x	const lvalue reference to the pair to be inserted.

Returns

std::pair<iterator, bool> returned by the _insert function.

3.2.2.17 insert() [2/2]

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
std::pair<iterator, bool> bst< value_type, key_type, cmp_op >::insert (
            pair_type && x )  [inline]
```

This function calls the _insert function using std::move() to insert a new node in the tree, if not already present.

Template Parameters

x	rvalue reference to the pair to be inserted.

Returns

std::pair<iterator, bool> returned by the _insert function.

3.2.2.18 operator=() [1/2]

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
bst& bst< value_type, key_type, cmp_op >::operator= (
            bst< value_type, key_type, cmp_op > && x )  [default], [noexcept]
```

Default move assignment for bst.

Template Parameters

x	rvalue reference to the tree.

3.2.2.19 operator=() [2/2]

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
bst& bst< value_type, key_type, cmp_op >::operator= (
            const bst< value_type, key_type, cmp_op > & x )  [inline]
```

Copy assignment for a bst tree.

Template Parameters

x	const lvalue reference to the tree to be copied.

Returns

lvalue reference to the copied tree.

3.2.2.20 operator[]() [1/2]

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
value_type& bst< value_type, key_type, cmp_op >::operator[] (
            const key_type & x )  [inline]
```

Overloaded operator that search the key to return corresponding associated value.

If the key is not present in the tree, it inserts a node with that key and as value a random value_type.

Template Parameters

x	const lvalue reference to key.

Returns

lvalue reference to the value mapped by the key.

3.2.2.21 operator[]() [2/2]

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
value_type& bst< value_type, key_type, cmp_op >::operator[] (
            key_type && x )  [inline]
```

Overloaded operator that search the key to return corresponding associated value.

If the key is not present in the tree, it inserts a node with that key and as value a random value_type.

Template Parameters

x	rvalue reference to key.

Returns

lvalue reference to the value mapped by the key.

3.2.3 Friends And Related Function Documentation

3.2.3.1 operator<<

```
template<typename value_type , typename key_type , typename cmp_op = std::less<key_type>>
std::ostream& operator<< (
            std::ostream & os,
            const bst< value_type, key_type, cmp_op > & x ) [friend]
```

Friend operator that prints the tree ordered by key values using const iterators.

Template Parameters

os	std::ostream& output stream object.
x	const lvalue reference to bst tree.

Returns

std::ostream& output stream object.

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

- bst.hpp

3.3 node< T > Struct Template Reference

```
#include <bst.hpp>
```

Collaboration diagram for node< T >:

Public Member Functions

- node (const T &x, node ∗p=nullptr)

  Constructor for node in which right and left are initialized to nullptr, parent to the input pointer to node and value to the input data x.
- node (const T &x, node ∗r, node ∗l, node ∗p=nullptr)

  Constructor for node in which right, left and parent are inizialized to input pointer and value to the input data x.
- node (T &&x, node ∗p=nullptr)

  Constructor for node in which right and left node are initialized to nullptr, parent to the input pointer to node and value to the input data x.
- node (T &&x, node ∗r, node ∗l, node ∗p=nullptr)

  Constructor for node in which right, left and parent are inizialized to input pointer and value to the input data x.
- node (const std::unique_ptr< node > &x, node ∗p=nullptr)

  Copy constructor for node, used by the copy constructor of bst.
- ∼node () noexcept=default

  Default deconstructor.

Public Attributes

- std::unique_ptr< node > right

  Unique pointer to the right node.
- std::unique_ptr< node > left

  Unique pointer to the left node.
- node * parent

  Raw pointer to the parent node.
- T value

  Member variable of type pair_type.

3.3.1 Detailed Description

template<typename T>
struct node< T >

Template Parameters

T	Template for an object of class pair_type.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 node() [1/5]

```
template<typename T >
node< T >::node (
              const T & x,
              node< T > * p = nullptr )  [inline], [explicit]
```

Constructor for node in which right and left are initialized to nullptr, parent to the input pointer to node and value to the input data x.

Template Parameters

x	const lvalue reference.
p	raw pointer to the parent node.

3.3.2.2 node() [2/5]

```
template<typename T >
node< T >::node (
              const T & x,
              node< T > * r,
              node< T > * l,
              node< T > * p = nullptr )   [inline]
```

Constructor for node in which right, left and parent are inizialized to input pointer and value to the input data x.

Template Parameters

x	const lvalue reference.
r	pointer to the right node.
l	pointer to the left node.
p	pointer to the parent node.

3.3.2.3 node() [3/5]

```
template<typename T >
node< T >::node (
                T && x,
                node< T > * p = nullptr )   [inline], [explicit]
```

Constructor for node in which right and left node are initialized to nullptr, parent to the input pointer to node and value to the input data x.

Template Parameters

x	rvalue reference.
p	raw pointer to the parent node.

3.3.2.4 node() [4/5]

```
template<typename T >
node< T >::node (
                T && x,
                node< T > * r,
                node< T > * l,
                node< T > * p = nullptr )   [inline]
```

Constructor for node in which right, left and parent are inizialized to input pointer and value to the input data x.

Template Parameters

x	rvalue reference.
r	pointer to the right node.
l	pointer to the left node.
p	pointer to the parent node.

3.3.2.5 node() [5/5]

```
template<typename T >
node< T >::node (
            const std::unique_ptr< node< T > > & x,
            node< T > * p = nullptr )  [inline], [explicit]
```

Copy constructor for node, used by the copy constructor of bst.

Set the parent with the input pointer, copies the value and recursively calls itself on the left and right node.

Template Parameters

x	std::unique_ptr to node to be copied.
p	pointer to the parent node.

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

- bst.hpp

Chapter 4

File Documentation

4.1 bst.hpp File Reference

Header containing the implementation of node struct and bst class.

```
#include <iostream>
#include <memory>
#include <utility>
#include <iterator>
#include <vector>
#include "iterator.hpp"
```

Include dependency graph for bst.hpp:

4.2 iterator.hpp File Reference

Header containing the implementation of _iterator class.

```
#include <iostream>
#include <memory>
#include <utility>
#include <iterator>
#include <vector>
#include "bst.hpp"
```

Include dependency graph for iterator.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class _iterator< node_t, T >

4.2.1 Detailed Description

Header containing the implementation of _iterator class.

Authors

Buscaroli Elena, Valeriani Lucrezia

4.3 main.cpp File Reference

Source code.

```
#include <iostream>
#include <memory>
#include <utility>
#include <iterator>
#include <vector>
#include "bst.hpp"
#include "iterator.hpp"
```

Include dependency graph for main.cpp:

Functions

- int **main** ()

4.3.1 Detailed Description

Source code.

Authors

Buscaroli Elena, Valeriani Lucrezia