Advanced Programming Exam - Binary Search Tree

Generated by Doxygen 1.8.13

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 NodeNamespace::BSTNode< K, T > Class Template Reference

Collaboration diagram for NodeNamespace::BSTNode< K, T >:

NodeNamespace::BSTNode < K, T >
parent

Public Member Functions

- BSTNode ()=default

  BSTNode default constructor.
- BSTNode (const std::pair< const K, T > &data)

  BSTNode constructor accepting a std::pair<K,T> as input.
- BSTNode (const std::pair< const K, T > &data, BSTNode ∗_parent)

  BSTNode constructor accepting a std::pair<K,T> and a pointer to the parent node as input.
- ∼BSTNode () noexcept=default

  Sefault BSTNode destructor.
- bool hasLChild () const noexcept

  Checks if the node has a left child.
- bool hasRChild () const noexcept

  Checks if the node has a right child.
- BSTNode ∗ get_next () noexcept

  Returns a pointer to the next node, according to key order.

Public Attributes

- std::pair< const K, T > **content**
- std::unique_ptr< BSTNode > **left**
- std::unique_ptr< BSTNode > **right**
- BSTNode ∗ **parent**

4.1.1 Member Function Documentation

4.1.1.1 get_next()

```
template<typename K , typename T >
NodeNamespace::BSTNode< K, T > * NodeNamespace::BSTNode< K, T >::get_next ( )  [noexcept]
```

Returns a pointer to the next node, according to key order.

Parameters

none	

Returns

Pointer to the next node.

4.1.1.2 hasLChild()

```
template<typename K, typename T>
bool NodeNamespace::BSTNode< K, T >::hasLChild ( ) const  [inline], [noexcept]
```

Checks if the node has a left child.

Parameters

none	

Returns

True if the node has a left child, False otherwise.

4.1.1.3 hasRChild()

```
template<typename K, typename T>
bool NodeNamespace::BSTNode< K, T >::hasRChild ( ) const  [inline], [noexcept]
```

Checks if the node has a right child.

Parameters

none	

Returns

True if the node has a right child, False otherwise.

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

- /home/w-wilson/DSSC/first_year/exam_advanced/test/include/BST.h
- /home/w-wilson/DSSC/first_year/exam_advanced/test/src/BST.hpp

4.2 BSTree< K, T, C > Class Template Reference

```
#include <BST.h>
```

Classes

- class ConstIterator
- class Iterator

Public Member Functions

- const int & size_of () const noexcept

  Returns the number of elements of the tree.
- bool is_empty () const noexcept

  Check if the tree is empty.
- const BSTNode< K, T > ∗ get_root () const

  Returns a pointer to the root node.
- BSTree ()=default

  Default BSTree constructor.
- BSTree (const K &key, const T &value, C comp=C{})

  BSTree constructor accepting a key and a value, inserted as root.
- BSTree (const std::pair< const K, T > &data, C comp=C{})

  BSTree constructor accepting a std::pair<key,value> inserted as root.
- ∼BSTree () noexcept=default

  Default BSTree destructor.
- BSTree (const BSTree &t)

  Copy constructor for a BSTree. Calls the auxiliary function copy_tree() which performs a deep copy of the tree.
- BSTree & operator= (const BSTree &t)

  Copy assignment of a BSTree.
- BSTree (BSTree< K, T, C > &&) noexcept=default

  Default move constructor for a BSTree.
- BSTree< K, T, C > & operator= (BSTree< K, T, C > &&) noexcept=default

Default move assignment for a BSTree.

- Iterator begin ()

  first element for iterating a BSTree.
- Iterator end ()

  last element for iterating a BSTree
- ConstIterator cbegin () const

  first element for iterating a BSTree.
- ConstIterator cend () const

  last element for iterating a BSTree
- bool insert (const std::pair< const K, T > &data)

  Inserts a BSTNode passing a std::pair<key,value>. Calls position_of(key) to find where the node should be appended. If the key is already present, does nothing. Otherwise a new node is created.
- bool insert (const K &key, const T &value)

  Inserts a node passing a key and a value separately.
- Iterator find (const K &key) const

  Returns, if found, an iterator to the node labelled by key.
- void clear () noexcept

  Wipes out the tree. Resets the root to nullptr, causing all the other nodes to be deleted. The tree is left uninitialized, but still usable.
- void print () const

  Prints the value in all the nodes, traversing the tree in order.
- void balance ()

  Balances the tree.
- T & operator[ ] (const K &k)

  Operator [] to access/insert a BSTNode. Returns the value associated with key if found. Otherwise, appends a node with the desired key and the default value.
- const T & **operator[ ]** (const K &k) const
- const T & square_bracket_test (const K &key) const

  Operator << to print a BSTree.

Public Attributes

- C **compare_f**

4.2.1 Detailed Description

template<typename K, typename T, typename C = std::less<K>>
class BSTree< K, T, C >

Binary Search Tree implementation.

Implementation of a Binary Search Tree. The tree is made of nodes which store a key/value pair. It requires a compare structure to order keys. If none is provided, it uses std::less<K>, where K is the key type.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 BSTree()

```
template<typename K, typename T, typename C = std::less<K>>
BSTree< K, T, C >::BSTree (
            const BSTree< K, T, C > & t )  [inline]
```

Copy constructor for a BSTree. Calls the auxiliary function copy_tree() which performs a deep copy of the tree.

Parameters

t	BSTree to be copied, passed by const reference.

4.2.3 Member Function Documentation

4.2.3.1 balance()

```
template<typename K , typename T , typename C >
void BSTree< K, T, C >::balance ( )
```

Balances the tree.

The tree is linearized into a vector of std::pair<key,value>. Then calls clear() and the auxiliary function balance(vector, begin, end).

Parameters

none	

Returns

none

4.2.3.2 begin()

```
template<typename K, typename T, typename C = std::less<K>>
Iterator BSTree< K, T, C >::begin ( ) [inline]
```

first element for iterating a BSTree.

Returns

iterator to the leftmost node.

4.2.3.3 cbegin()

```
template<typename K, typename T, typename C = std::less<K>>
ConstIterator BSTree< K, T, C >::cbegin ( ) const [inline]
```

first element for iterating a BSTree.

Returns

constiterator to the leftmost node.

4.2.3.4 cend()

```
template<typename K, typename T, typename C = std::less<K>>
ConstIterator BSTree< K, T, C >::cend ( ) const  [inline]
```

last element for iterating a BSTree

Returns

cosntiterator to nullptr.

4.2.3.5 clear()

```
template<typename K , typename T , typename C >
void BSTree< K, T, C >::clear ( )  [noexcept]
```

Wipes out the tree. Resets the root to nullptr, causing all the other nodes to be deleted. The tree is left uninitialized, but still usable.

Parameters

none	

Returns

none

4.2.3.6 end()

```
template<typename K, typename T, typename C = std::less<K>>
Iterator BSTree< K, T, C >::end ( )  [inline]
```

last element for iterating a BSTree

Returns

iterator to nullptr.

4.2.3.7 find()

```
template<typename K , typename T , typename C >
BSTree< K, T, C >::Iterator BSTree< K, T, C >::find (
             const K & key ) const
```

Returns, if found, an iterator to the node labelled by key.

Calls the auxiliary function position_of(key) which returns, if key is found, an iterator pointing to the node that contains key. If the key is not found or the BSTree is empty, retuns a ConstIterator to nullptr.

Parameters

key	Key to be found.

Returns

[Iterator](#) to the node containing the requested key or to nullptr.

4.2.3.8 get_root()

```
template<typename K, typename T, typename C = std::less<K>>
const BSTNode<K,T>* BSTree< K, T, C >::get_root ( ) const  [inline]
```

Returns a pointer to the root node.

Parameters

none	

Returns

Pointer to root BSTNode.

4.2.3.9 insert() [1/2]

```
template<typename K , typename T , typename C >
bool BSTree< K, T, C >::insert (
             const std::pair< const K, T > & data )
```

Inserts a BSTNode passing a std::pair<key,value>. Calls position_of(key) to find where the node should be appended. If the key is already present, does nothing. Otherwise a new node is created.

Parameters

data	Pair of key-value.

Returns

bool Boolean which is true if a new node was inserted, false if the key was already present.

4.2.3.10 insert() [2/2]

```
template<typename K , typename T , typename C >
bool BSTree< K, T, C >::insert (
             const K & key,
             const T & value )
```

Inserts a node passing a key and a value separately.

Key and value are grouped in an std::pair<key, value>, insert() is then called passing the std::pair<key,value>.

Parameters

key	key entry of the node.
value	content associated to key.

Returns

bool

4.2.3.11 is_empty()

```
template<typename K, typename T, typename C = std::less<K>>
bool BSTree< K, T, C >::is_empty ( ) const  [inline], [noexcept]
```

Check if the tree is empty.

Parameters

none	

Returns

True if the tree is empty, false otherwise.

4.2.3.12 operator=()

```
template<typename K , typename T , typename C >
BSTree< K, T, C > & BSTree< K, T, C >::operator= (
             const BSTree< K, T, C > & t )
```

Copy assignment of a BSTree.

Parameters

t	the BSTree to be copied.

Returns

The newly copied BSTree.

4.2.3.13 operator[]()

```
template<typename K , typename T , typename C >
T & BSTree< K, T, C >::operator[] (
            const K & k )
```

Operator [] to access/insert a BSTNode. Returns the value associated with key if found. Otherwise, appends a node with the desired key and the default value.

Parameters

k	key to be found/inserted.

Returns

the value associated with the key k.

4.2.3.14 print()

```
template<typename K , typename T , typename C >
void BSTree< K, T, C >::print ( ) const
```

Prints the value in all the nodes, traversing the tree in order.

Parameters

none	

Returns

none

4.2.3.15 size_of()

```
template<typename K, typename T, typename C = std::less<K>>
const int& BSTree< K, T, C >::size_of ( ) const  [inline], [noexcept]
```

Returns the number of elements of the tree.

Parameters

none	

Returns

const int& Number of elements of the tree.

4.2.3.16 square_bracket_test()

```
template<typename K , typename T , typename C >
const T & BSTree< K, T, C >::square_bracket_test (
             const K & key ) const
```

Operator << to print a BSTree.

Iterates the tree using const iterators and prints "key: value" for each node.

Parameters

os	OutputStream, where the content of the BSTree should be printed.
t	Const reference to the BSTree to be printed.

Returns

Reference to the OutputStream.

Test function for const[] operator

Trivial function declared as const which calls the const[] operator

Parameters

key	key to be searched

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

- /home/w-wilson/DSSC/first_year/exam_advanced/test/include/BST.h
- /home/w-wilson/DSSC/first_year/exam_advanced/test/src/BST.hpp
- /home/w-wilson/DSSC/first_year/exam_advanced/test/src/main.cc

4.3 compareh Struct Reference

Public Member Functions

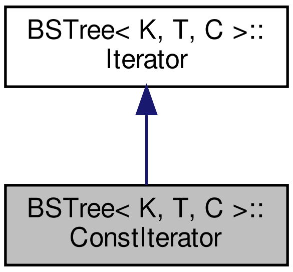
- bool **operator()** (RandomKey_explicit bingo, RandomKey_explicit bango) const

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

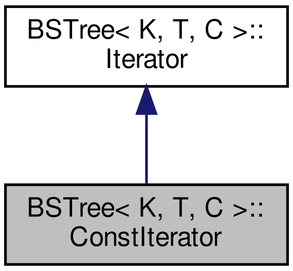
- /home/w-wilson/DSSC/first_year/exam_advanced/test/src/main.cc

4.4 BSTree< K, T, C >::ConstIterator Class Reference

Inheritance diagram for BSTree< K, T, C >::ConstIterator:



Collaboration diagram for BSTree< K, T, C >::ConstIterator:



Public Types

- using **parent** = BSTree< K, T, C >::Iterator

Public Member Functions

- const std::pair< const K, T > & **operator*** () const

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

- /home/w-wilson/DSSC/first_year/exam_advanced/test/src/BST.hpp

4.5 error Struct Reference

Public Member Functions

- **error** (const std::string &s) noexcept

Public Attributes

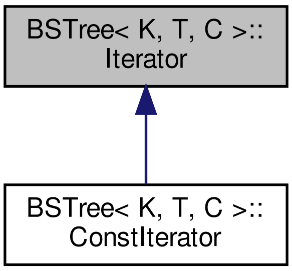
- std::string **message**

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

- /home/w-wilson/DSSC/first_year/exam_advanced/test/src/BST.hpp

4.6 BSTree$<$ K, T, C $>$::Iterator Class Reference

Inheritance diagram for BSTree$<$ K, T, C $>$::Iterator:



Public Member Functions

- **Iterator** (BSTNode$<$ K, T $>$ $*$n)
- std::pair$<$ const K, T $>$ & **operator$*$** () const
- BSTNode$<$ K, T $>$ $*$ **get** ()
- Iterator & **operator++** () noexcept
- bool **operator==** (const Iterator &other) const noexcept
- bool **operator!=** (const Iterator &other) const noexcept

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

- /home/w-wilson/DSSC/first_year/exam_advanced/test/src/BST.hpp

4.7 RandomKey Struct Reference

Public Member Functions

- bool **operator**$<$ (RandomKey other) const

Public Attributes

- int **one**
- int **two**
- int **three**

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

- /home/w-wilson/DSSC/first_year/exam_advanced/test/src/main.cc

4.8 RandomKey_explicit Struct Reference

Public Attributes

- int **one**
- int **two**
- int **three**

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

- /home/w-wilson/DSSC/first_year/exam_advanced/test/src/main.cc

Chapter 5

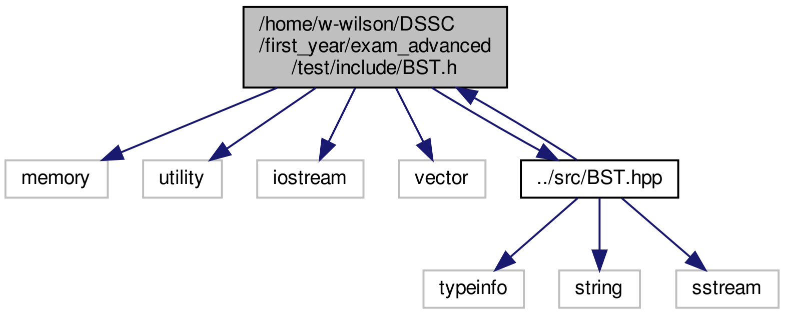
File Documentation

5.1 /home/w-wilson/DSSC/first_year/exam_advanced/test/include/BST.h File Reference

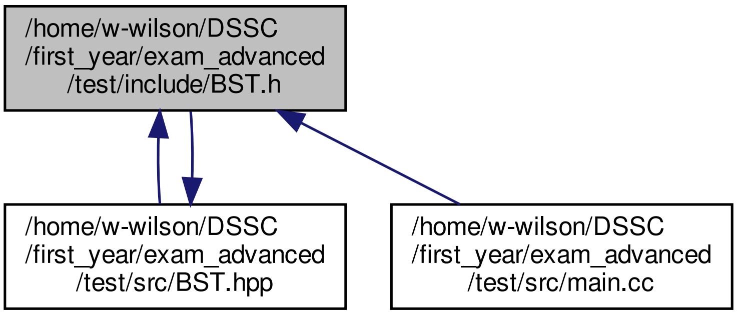
BST header file.

```
#include <memory>
#include <utility>
#include <iostream>
#include <vector>
#include "../src/BST.hpp"
```

Include dependency graph for BST.h:


/home/w-wilson/DSSC/first_year/exam_advanced/test/include/BST.h
memory
utility
iostream
vector
../src/BST.hpp
typeinfo
string
sstream

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



Classes

- class NodeNamespace::BSTNode< K, T >
- class BSTree< K, T, C >

Typedefs

- template<typename K , typename T >
  using **BSTNode** = NodeNamespace::BSTNode< K, T >

5.1.1 Detailed Description

BST header file.

Author

Amadio Simone, Indri Patrick

Date

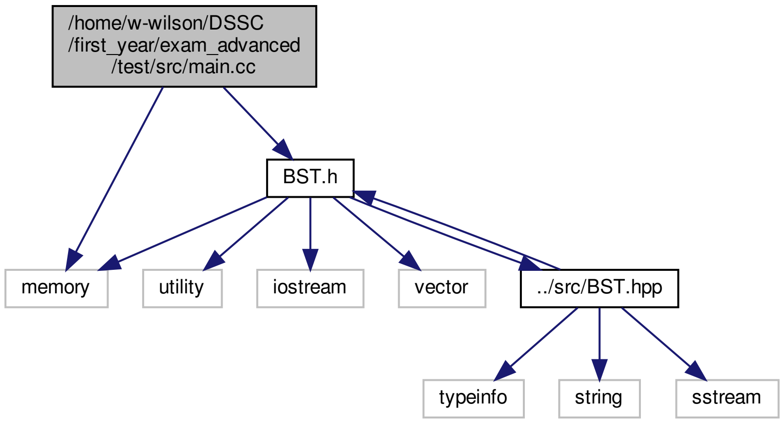
25/01/19

5.2 /home/w-wilson/DSSC/first_year/exam_advanced/test/src/main.cc File Reference

main file for the exam

```
#include <memory>
#include "BST.h"
```

Include dependency graph for main.cc:



Classes

- struct RandomKey_explicit
- struct compareh
- struct RandomKey

Functions

- std::ostream & **operator**<< (std::ostream &os, const RandomKey &k)
- int **main** ()

5.2.1 Detailed Description

main file for the exam

Author

William WIlson

Date

1/2/23

Index