# Advance Programming Exam: bst with smart pointers.

Generated by Doxygen 1.8.17

# Chapter 1

# Binary Search Tree

Lecturer Dr. Alberto Sartori

**Author**

Valentin Nkana

**Date**

July 2021

## 1.1 Introduction

This project was implemented as a final exam for C++ in the course of Advanced Programming.

## 1.2 Compilation

The compilation of the files is done by running the 'make' command; use the command 'make clean' to remove object files and executables. The executable 'benchmark.x' requires an argument from command line representing the number of nodes in the tree and std::map : e.g we can run './benchmark.o 50' will run the tests for a tree and std::map with 50 nodes of (random int keys and int values). The corresponding output will be three sets of 11 time measurements for the find() method (for unbalanced tree, balanced tree, std::map each).

# 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 files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 bst< K, V, C >::__iterator< O > Class Template Reference

Iterator definition of the class bst.

`#include <bst.hpp>`

### Public Types

- using value_type = O
- using difference_type = std::ptrdiff_t
- using iterator_category = std::forward_iterator_tag
- using reference = value_type &
- using pointer = value_type *

### Public Member Functions

- __iterator (Node< K, V > *x) noexcept
  *Custom constructor for iterator.*
- __iterator () noexcept=default
  *Default-generated constructor.*
- ~__iterator ()=default
  *Default-generated destructor.*
- reference operator* () const noexcept
  *Dereference operator.*
- pointer operator-> () const noexcept
- __iterator & operator++ () noexcept
- __iterator operator++ (int)
  *Post increment operator.*
- void print_iterator ()

### Friends

- class bst
- bool operator== (const __iterator &a, const __iterator &b)
- bool operator!= (const __iterator &a, const __iterator &b)

### 4.1.1 Detailed Description

**template<typename K, typename V, typename C = std::less<K>>**
**template<typename O>**
**class bst< K, V, C >::__iterator< O >**

Iterator definition of the class bst.

### 4.1.2 Member Typedef Documentation

#### 4.1.2.1 difference_type

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
using bst< K, V, C >::__iterator< O >::difference_type = std::ptrdiff_t
```

#### 4.1.2.2 iterator_category

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
using bst< K, V, C >::__iterator< O >::iterator_category = std::forward_iterator_tag
```

#### 4.1.2.3 pointer

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
using bst< K, V, C >::__iterator< O >::pointer = value_type *
```

#### 4.1.2.4 reference

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
using bst< K, V, C >::__iterator< O >::reference = value_type &
```

#### 4.1.2.5 value_type

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
using bst< K, V, C >::__iterator< O >::value_type = O
```

### 4.1.3 Constructor & Destructor Documentation

#### 4.1.3.1 __iterator() [1/2]

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
bst< K, V, C >::__iterator< O >::__iterator (
            Node< K, V > * x )  [inline], [explicit], [noexcept]
```

Custom constructor for iterator.

**Parameters**

| | |
|---|---|
| *x* | Raw pointer to a Node |

Construct a new __iterator that refers to Node x

#### 4.1.3.2 __iterator() [2/2]

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
bst< K, V, C >::__iterator< O >::__iterator ( )  [default], [noexcept]
```

Default-generated constructor.

#### 4.1.3.3 ~__iterator()

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
bst< K, V, C >::__iterator< O >::~__iterator ( )  [default]
```

Default-generated destructor.

### 4.1.4 Member Function Documentation

#### 4.1.4.1 operator∗()

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
reference bst< K, V, C >::__iterator< O >::operator* ( ) const  [inline], [noexcept]
```

Dereference operator.

Dereferences an __iterator by returning the data stored by the Node it refers to

#### 4.1.4.2 operator++() [1/2]

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
__iterator& bst< K, V, C >::__iterator< O >::operator++ ( )  [inline], [noexcept]
```

#### 4.1.4.3 operator++() [2/2]

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
__iterator bst< K, V, C >::__iterator< O >::operator++ (
            int  )  [inline]
```

Post increment operator.

**Returns**

__iterator

#### 4.1.4.4 operator->()

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
pointer bst< K, V, C >::__iterator< O >::operator-> ( ) const  [inline], [noexcept]
```

#### 4.1.4.5 print_iterator()

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
void bst< K, V, C >::__iterator< O >::print_iterator ( )  [inline]
```

### 4.1.5 Friends And Related Function Documentation

#### 4.1.5.1 bst

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
friend class bst  [friend]
```

#### 4.1.5.2 operator"!=

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
bool operator!= (
            const __iterator< O > & a,
            const __iterator< O > & b )  [friend]
```

#### 4.1.5.3 operator==

```
template<typename K , typename V , typename C = std::less<K>>
template<typename O >
bool operator== (
            const __iterator< O > & a,
            const __iterator< O > & b )  [friend]
```

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

- /home/valentinnkana/Documents/Advance_Programing_Exam/bst.hpp

## 4.2 bst< K, V, C > Class Template Reference

declaration of the members and methods of bst class

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

### Classes

- class __iterator

  *Iterator definition of the class bst.*

## Public Types

- using iterator = __iterator< std::pair< const K, V > >
- using const_iterator = __iterator< const std::pair< const K, V > >

## Public Member Functions

- bst ()
- bst (bst &&tree) noexcept
- bst & operator= (bst &&tree) noexcept
- bst (const bst &tree)
- bst & operator= (bst &tree)
- Node< K, V > * root () noexcept
- void clear () noexcept
- void erase (const K &x)
- template<typename T >
  V & operator[ ] (T &&key)

  *Overloadding of the operator [].*
- iterator begin () noexcept
- iterator end () noexcept
- const_iterator begin () const noexcept
- const_iterator end () const noexcept
- const_iterator cbegin () const noexcept
- const_iterator cend () const noexcept
- iterator find (const K &key)
- const_iterator find (const K &key) const

  *To find an node inside the binary tree.*
- std::pair< iterator, bool > insert (const std::pair< const K, V > &data)

  *To insert a node inside the bst first version.*
- std::pair< iterator, bool > insert (std::pair< const K, V > &&data)
- template<typename... Types>
  std::pair< iterator, bool > emplace (Types &&...args)
- void balance ()
- bool is_balanced ()
- template<typename... Types>
  std::pair< typename bst< K, V, C >::iterator, bool > emplace (Types &&...args)

  *definition of the emplace method*

## Friends

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

### 4.2.1 Detailed Description

**template<typename K, typename V, typename C = std::less<K>>**
**class bst< K, V, C >**

declaration of the members and methods of bst class

## 4.2.2 Member Typedef Documentation

### 4.2.2.1 const_iterator

```
template<typename K , typename V , typename C = std::less<K>>
using bst< K, V, C >::const_iterator = __iterator<const std::pair<const K, V> >
```

### 4.2.2.2 iterator

```
template<typename K , typename V , typename C = std::less<K>>
using bst< K, V, C >::iterator = __iterator<std::pair<const K, V> >
```

## 4.2.3 Constructor & Destructor Documentation

### 4.2.3.1 bst() [1/3]

```
template<typename K , typename V , typename C = std::less<K>>
bst< K, V, C >::bst ( )  [inline]
```

### 4.2.3.2 bst() [2/3]

```
template<typename K , typename V , typename C = std::less<K>>
bst< K, V, C >::bst (
            bst< K, V, C > && tree )  [inline], [noexcept]
```

### 4.2.3.3 bst() [3/3]

```
template<typename K , typename V , typename C = std::less<K>>
bst< K, V, C >::bst (
            const bst< K, V, C > & tree )  [inline], [explicit]
```

## 4.2.4 Member Function Documentation

#### 4.2.4.1 balance()

```
template<typename K , typename V , typename C >
void bst< K, V, C >::balance
```

#### 4.2.4.2 begin() [1/2]

```
template<typename K , typename V , typename C = std::less<K>>
const_iterator bst< K, V, C >::begin ( ) const  [inline], [noexcept]
```

#### 4.2.4.3 begin() [2/2]

```
template<typename K , typename V , typename C = std::less<K>>
iterator bst< K, V, C >::begin ( )  [inline], [noexcept]
```

#### 4.2.4.4 cbegin()

```
template<typename K , typename V , typename C = std::less<K>>
const_iterator bst< K, V, C >::cbegin ( ) const  [inline], [noexcept]
```

#### 4.2.4.5 cend()

```
template<typename K , typename V , typename C = std::less<K>>
const_iterator bst< K, V, C >::cend ( ) const  [inline], [noexcept]
```

#### 4.2.4.6 clear()

```
template<typename K , typename V , typename C = std::less<K>>
void bst< K, V, C >::clear ( )  [inline], [noexcept]
```

#### 4.2.4.7 emplace() [1/2]

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

#### 4.2.4.8 emplace() [2/2]

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

definition of the emplace method

#### 4.2.4.9 end() [1/2]

```
template<typename K , typename V , typename C = std::less<K>>
const_iterator bst< K, V, C >::end ( ) const  [inline], [noexcept]
```

#### 4.2.4.10 end() [2/2]

```
template<typename K , typename V , typename C = std::less<K>>
iterator bst< K, V, C >::end ( )  [inline], [noexcept]
```

#### 4.2.4.11 erase()

```
template<typename K , typename V , typename C = std::less<K>>
void bst< K, V, C >::erase (
            const K & x )  [inline]
```

#### 4.2.4.12 find() [1/2]

```
template<typename K , typename V , typename C >
bst< K, V, C >::iterator bst< K, V, C >::find (
            const K & key )
```

#### 4.2.4.13 find() [2/2]

```
template<typename K , typename V , typename C >
bst< K, V, C >::const_iterator bst< K, V, C >::find (
            const K & key ) const
```

To find an node inside the binary tree.

#### 4.2.4.14 insert() [1/2]

```
template<typename K , typename V , typename C >
std::pair< typename bst< K, V, C >::iterator, bool > bst< K, V, C >::insert (
            const std::pair< const K, V > & data )
```

To insert a node inside the bst first version.

#### 4.2.4.15 insert() [2/2]

```
template<typename K , typename V , typename C >
std::pair< typename bst< K, V, C >::iterator, bool > bst< K, V, C >::insert (
            std::pair< const K, V > && data )
```

#### 4.2.4.16 is_balanced()

```
template<typename K , typename V , typename C = std::less<K>>
bool bst< K, V, C >::is_balanced ( )  [inline]
```

#### 4.2.4.17 operator=() [1/2]

```
template<typename K , typename V , typename C = std::less<K>>
bst& bst< K, V, C >::operator= (
            bst< K, V, C > && tree )  [inline], [noexcept]
```

#### 4.2.4.18 operator=() [2/2]

```
template<typename K , typename V , typename C = std::less<K>>
bst& bst< K, V, C >::operator= (
            bst< K, V, C > & tree )  [inline]
```

#### 4.2.4.19 operator[]()

```
template<typename K , typename V , typename C >
template<typename T >
V & bst< K, V, C >::operator[] (
            T && key )
```

Overloadding of the operator [].

#### 4.2.4.20 root()

```
template<typename K , typename V , typename C = std::less<K>>
Node<K, V>* bst< K, V, C >::root ( )  [inline], [noexcept]
```

### 4.2.5 Friends And Related Function Documentation

#### 4.2.5.1 operator<<

```
template<typename K , typename V , typename C = std::less<K>>
std::ostream& operator<< (
            std::ostream & os,
            const bst< K, V, C > & tree )  [friend]
```

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

- /home/valentinnkana/Documents/Advance_Programing_Exam/bst.hpp

## 4.3 Node< K, V > Struct Template Reference

Definition and declaration of the object Node.

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

Collaboration diagram for Node< K, V >:

### Public Member Functions

- Node (const std::pair< const K, V > &data, Node< K, V > *left, Node< K, V > *right, Node< K, V > *parent) noexcept
- Node (const std::pair< const K, V > &data) noexcept
- Node (const std::pair< const K, V > &data, Node< K, V > *parent) noexcept
- Node ()=default
- Node (std::pair< const K, V > &&data, Node< K, V > *left, Node< K, V > *right, Node< K, V > *parent) noexcept
- Node (std::pair< const K, V > &&data) noexcept
- ~Node ()=default
- Node (std::pair< const K, V > &&data, Node< K, V > *parent) noexcept
- Node (const std::unique_ptr< Node< K, V >> &pn, Node< K, V > *parent)
- const K key () noexcept
- V value () noexcept
- void to_print ()

## Public Attributes

- std::unique_ptr< Node > _left
- std::unique_ptr< Node > _right
- Node * _parent
- std::pair< const K, V > _data

### 4.3.1 Detailed Description

**template<typename K, typename V>**
**struct Node< K, V >**

Definition and declaration of the object Node.

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 Node() [1/8]

```
template<typename K , typename V >
Node< K, V >::Node (
            const std::pair< const K, V > & data,
            Node< K, V > * left,
            Node< K, V > * right,
            Node< K, V > * parent )  [inline], [noexcept]
```

#### 4.3.2.2 Node() [2/8]

```
template<typename K , typename V >
Node< K, V >::Node (
            const std::pair< const K, V > & data )  [inline], [explicit], [noexcept]
```

#### 4.3.2.3 Node() [3/8]

```
template<typename K , typename V >
Node< K, V >::Node (
            const std::pair< const K, V > & data,
            Node< K, V > * parent )  [inline], [noexcept]
```

#### 4.3.2.4 Node() [4/8]

```
template<typename K , typename V >
Node< K, V >::Node ( )  [default]
```

#### 4.3.2.5 Node() [5/8]

```
template<typename K , typename V >
Node< K, V >::Node (
            std::pair< const K, V > && data,
            Node< K, V > * left,
            Node< K, V > * right,
            Node< K, V > * parent )  [inline], [noexcept]
```

#### 4.3.2.6 Node() [6/8]

```
template<typename K , typename V >
Node< K, V >::Node (
            std::pair< const K, V > && data )  [inline], [noexcept]
```

#### 4.3.2.7 ~Node()

```
template<typename K , typename V >
Node< K, V >::~Node ( )  [default]
```

#### 4.3.2.8 Node() [7/8]

```
template<typename K , typename V >
Node< K, V >::Node (
            std::pair< const K, V > && data,
            Node< K, V > * parent )  [inline], [noexcept]
```

#### 4.3.2.9 Node() [8/8]

```
template<typename K , typename V >
Node< K, V >::Node (
            const std::unique_ptr< Node< K, V >> & pn,
            Node< K, V > * parent )  [inline]
```

### 4.3.3 Member Function Documentation

#### 4.3.3.1 key()

```
template<typename K , typename V >
const K Node< K, V >::key ( )  [inline], [noexcept]
```

#### 4.3.3.2 to_print()

```
template<typename K , typename V >
void Node< K, V >::to_print ( )  [inline]
```

#### 4.3.3.3 value()

```
template<typename K , typename V >
V Node< K, V >::value ( )  [inline], [noexcept]
```

### 4.3.4 Member Data Documentation

#### 4.3.4.1 _data

```
template<typename K , typename V >
std::pair<const K, V> Node< K, V >::_data
```

#### 4.3.4.2 _left

```
template<typename K , typename V >
std::unique_ptr<Node> Node< K, V >::_left
```

#### 4.3.4.3 _parent

```
template<typename K , typename V >
Node* Node< K, V >::_parent
```

#### 4.3.4.4 _right

```
template<typename K , typename V >
std::unique_ptr<Node> Node< K, V >::_right
```

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

- /home/valentinnkana/Documents/Advance_Programing_Exam/bst.hpp

# Chapter 5

# File Documentation

## 5.1 /home/valentinnkana/Documents/Advance_Programing_↩ Exam/benchmark.cpp File Reference

```
#include "bst.hpp"
#include <cstdlib>
#include <iostream>
#include <ctime>
#include <chrono>
#include <map>
```

Include dependency graph for benchmark.cpp:

## 5.2 /home/valentinnkana/Documents/Advance_Programing_↩ Exam/bst.hpp File Reference

```
#include <iostream>
#include <algorithm>
#include <utility>
#include <iterator>
#include <memory>
#include <functional>
#include <vector>
```

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

### Classes

- struct Node< K, V >

  *Definition and declaration of the object Node.*
- class bst< K, V, C >

  *declaration of the members and methods of bst class*
- class bst< K, V, C >::__iterator< O >

  *Iterator definition of the class bst.*
- class bst< K, V, C >::__iterator< O >

  *Iterator definition of the class bst.*

## 5.3 /home/valentinnkana/Documents/Advance_Programing_Exam/test_bst_tree.cpp File Reference

```
#include "bst.hpp"
#include <cstdlib>
#include <iostream>
#include <ctime>
#include <chrono>
#include <map>
```

Include dependency graph for test_bst_tree.cpp:

### Functions

- int main ()

### 5.3.1 Function Documentation

#### 5.3.1.1 main()

```
int main ( )
```

# Index