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

Class Documentation

2.1 AP_it::__iterator< node, O > Class Template Reference

```
#include <iterator.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 *x) noexcept
- reference operator* () const noexcept
- pointer operator-> () const noexcept
- __iterator & operator++ () noexcept
- __iterator operator++ (int) noexcept

Friends

- template<typename K , typename V , typename cmp >
  class **BST::bst**
- bool operator== (const __iterator &a, const __iterator &b)
- bool operator!= (const __iterator &a, const __iterator &b)

2.1.1 Detailed Description

template<typename node, typename O>
class AP_it::__iterator< node, O >

Iterator class for the bst_tree with nodes having left and right children. The iterator has a pointer to a node called current. The aim of these iterators is to move easily following the nodes

2.1.2 Constructor & Destructor Documentation

2.1.2.1 __iterator()

```
template<typename node, typename O>
AP_it::__iterator< node, O >::__iterator (
             node * x ) [inline], [explicit], [noexcept]
```

Explicit constructor of iterator, it initializes its temporary node to given a node

2.1.3 Member Function Documentation

2.1.3.1 operator*()

```
template<typename node, typename O>
reference AP_it::__iterator< node, O >::operator* ( ) const [inline], [noexcept]
```

Dereferent operator, it dereference the iterator returning the value of the node the iterator is pointing to

2.1.3.2 operator++() [1/2]

```
template<typename node, typename O>
__iterator& AP_it::__iterator< node, O >::operator++ ( ) [inline], [noexcept]
```

pre-increment operator ++, it easily moves from a node to the in order successor node along the tree

2.1.3.3 operator++() [2/2]

```
template<typename node, typename O>
__iterator AP_it::__iterator< node, O >::operator++ (
             int ) [inline], [noexcept]
```

post-increment operator ++, create a temporary iterator to the current node and then increase current with pre-increment

2.1.3.4 operator->()

```
template<typename node, typename O>
pointer AP_it::__iterator< node, O >::operator-> ( ) const [inline], [noexcept]
```

Arrow operator, it returns a pointer to the value of the node the iterator is pointing to

2.1.4 Friends And Related Function Documentation

2.1.4.1 operator"!=

```
template<typename node, typename O>
bool operator!= (
            const __iterator< node, O > & a,
            const __iterator< node, O > & b )  [friend]
```

not equal

2.1.4.2 operator==

```
template<typename node, typename O>
bool operator== (
            const __iterator< node, O > & a,
            const __iterator< node, O > & b )  [friend]
```

two iterators are equal if they point to the same node

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

- include/iterator.hpp

2.2 BST::bst< K, V, cmp > Class Template Reference

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

Public Member Functions

- bst () noexcept
- bst (bst &&) noexcept=default
- bst & operator= (bst &&) noexcept=default
- bst (const bst &) noexcept
- bst & operator= (const bst &tree)
- template<class... Types>
  std::pair< iterator, bool > emplace (Types &&... args)
- template<class OT >
  std::pair< iterator, bool > insert (OT &&x)
- iterator begin () noexcept
- const_iterator begin () const noexcept
- const_iterator cbegin () const noexcept
- iterator end () noexcept
- const_iterator end () const noexcept
- const_iterator cend () const noexcept
- iterator find (const K &x)
- const_iterator find (const K &x) const
- void clear ()
- void erase (const K &x)
- void balance ()
- void sortedArrayToBST (std::vector< pair_type > data, int start, int end, bst &balanced_tree)
- template<class OT >
  V & operator[ ] (OT &&x)
- template<class OT >
  std::pair< AP_it::__iterator< AP_node::node< std::pair< const K, V > >, std::pair< const K, V > >, bool > **insert** (OT &&x)

Friends

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

2.2.1 Detailed Description

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

Binary Search Tree class, templated on the values to be assigned to the nodes and on the comprison method In this particular case, the data is initialised to a std::pair<key_type =K, value_typr=V> and the comparison method is chosen to be std::less<key_type>. Most of the implementations are written in src/bst.cc

2.2.2 Constructor & Destructor Documentation

2.2.2.1 bst() [1/3]

```
template<class K , class V , class cmp = std::less<K>>
BST::bst< K, V, cmp >::bst ( )  [inline], [explicit], [noexcept]
```

Default (empty) constructor of a BST; it initialize the comparison method to default and the head to nullptr

2.2.2.2 bst() [2/3]

```
template<class K , class V , class cmp = std::less<K>>
BST::bst< K, V, cmp >::bst (
             bst< K, V, cmp > &&  )  [explicit], [default], [noexcept]
```

Default move constructor of a BST

2.2.2.3 bst() [3/3]

```
template<class K , class V , class cmp >
BST::bst< K, V, cmp >::bst (
             const bst< K, V, cmp > & tree )  [explicit], [noexcept]
```

Copy constructor of a BST

2.2.3 Member Function Documentation

2.2.3.1 balance()

```
template<class K , class V , class cmp >
void BST::bst< K, V, cmp >::balance ( )
```

balances the the bst by successively inserting the median of the odered sub-vectors associated with the tree. See src/bst.cc implementation for a step-by-step explanation

2.2.3.2 begin() [1/2]

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

returns an iterator pointing to the first node

2.2.3.3 begin() [2/2]

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

returns a constant iterator pointing to the first node

2.2.3.4 cbegin()

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

returns a constant iterator pointing to the first node

2.2.3.5 cend()

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

returns a constant iterator pointing to the node found with __end()

2.2.3.6 clear()

```
template<class K , class V , class cmp = std::less<K>>
void BST::bst< K, V, cmp >::clear ( )  [inline]
```

clears the whole bst by resetting the head to nullptr

2.2.3.7 emplace()

```
template<class K , class V , class cmp = std::less<K>>
template<class...  Types>
std::pair<iterator,bool> BST::bst< K, V, cmp >::emplace (
             Types &&...  args )  [inline]
```

Emplace function, inserts a node with the simple expression emplace(a,b) while insert needs an explicit pair_type

2.2.3.8 end() [1/2]

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

returns an iterator pointing to the node found with __end()

2.2.3.9 end() [2/2]

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

returns a constant iterator pointing to the node found with __end()

2.2.3.10 erase()

```
template<class K , class V , class cmp >
void BST::bst< K, V, cmp >::erase (
             const K & x )
```

erases the node having the key x if present; is such a case the tree needs to be "jointed" back in order not to lose the parent-child relationships see src/bst.cc implementation for a step-by-step explanation

2.2.3.11 find() [1/2]

```
template<class K , class V , class cmp >
AP_it::__iterator< AP_node::node< std::pair< const K, V > >, std::pair< const K, V > > BS↩
T::bst< K, V, cmp >::find (
             const K & x )
```

navigate from the head using the op operator and stopping if the key is found. If key x is present, it returns the iterator pointing to the corresponding node; otherwise ir returns nullptr

2.2.3.12 find() [2/2]

```
template<class K , class V , class cmp >
AP_it::__iterator< AP_node::node< std::pair< const K, V > >, const std::pair< const K, V >
> BST::bst< K, V, cmp >::find (
             const K & x ) const
```

const interator version of find() funxtion

2.2.3.13 insert()

```
template<class K , class V , class cmp = std::less<K>>
template<class OT >
std::pair<iterator, bool> BST::bst< K, V, cmp >::insert (
             OT && x )
```

insert operator accepts both rvalue and lvalue variables thanks to the extra template OT. It returns a pair made up by the iterator pointing to the inserted node and a bool which is true if the key has been inserted or false if it was already present

2.2.3.14 operator=() [1/2]

```
template<class K , class V , class cmp = std::less<K>>
bst& BST::bst< K, V, cmp >::operator= (
            bst< K, V, cmp > &&  )  [default], [noexcept]
```

Default move assignement of a BST

2.2.3.15 operator=() [2/2]

```
template<class K , class V , class cmp = std::less<K>>
bst& BST::bst< K, V, cmp >::operator= (
            const bst< K, V, cmp > & tree )  [inline]
```

Copy assignement

2.2.3.16 operator[]()

```
template<class K , class V , class cmp = std::less<K>>
template<class OT >
V& BST::bst< K, V, cmp >::operator[] (
            OT && x )  [inline]
```

put-to operator, returs the value corresponding to such key if present; otherwise it adds the node and set the value to zero or to the rhs_value, according to bst[key] = rhs_value. It can act both on rvalue and lvalue thanks to the extra template OT

2.2.3.17 sortedArrayToBST()

```
template<class K , class V , class cmp = std::less<K>>
void BST::bst< K, V, cmp >::sortedArrayToBST (
            std::vector< pair_type > data,
            int start,
            int end,
            bst< K, V, cmp > & balanced_tree )
```

helping function for balance()

2.2.4 Friends And Related Function Documentation

2.2.4.1 operator<<

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

stream operator, prints each node of the tree from the first to the last node according to the comparison method

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

- include/bst.hpp
- src/bst.cc

2.3 AP_node::node< T > Struct Template Reference

```
#include <node.hpp>
```

Public Member Functions

- node () noexcept
- node (const T &v, node< T > ∗pu) noexcept
- node (T &&v, node< T > ∗pu) noexcept

Public Attributes

- T value
- std::unique_ptr< node< T > > left
- std::unique_ptr< node< T > > right
- node< T > ∗ upper

2.3.1 Detailed Description

template<class T>
struct AP_node::node< T >

Definition and declaration of the templated node structure which will be inserted in the Binary Tree structure. The nodes have a value and they have three pointers pointing to right/left children and parent.

2.3.2 Constructor & Destructor Documentation

2.3.2.1 node() [1/3]

```
template<class T>
AP_node::node< T >::node ( )  [inline], [explicit], [noexcept]
```

standard constructor setting the value to zero and to nullptr all the pointers

2.3.2.2 node() [2/3]

```
template<class T>
AP_node::node< T >::node (
             const T & v,
             node< T > * pu )  [inline], [explicit], [noexcept]
```

custom constructor copying the value and setting only the upper pointer. Used whenever a left-value is given for the insertion in the tree

2.3.2.3 node() [3/3]

```
template<class T>
AP_node::node< T >::node (
             T && v,
             node< T > * pu )  [inline], [explicit], [noexcept]
```

custom move constructor, moving the value to the node and setting only the upper pointer. Used whenever a right-value is given for the insertion in the tree

2.3.3 Member Data Documentation

2.3.3.1 left

```
template<class T>
std::unique_ptr<node<T> > AP_node::node< T >::left
```

unique pointer to the left child (which is clearly a node templated on the value type)

2.3.3.2 right

```
template<class T>
std::unique_ptr<node<T> > AP_node::node< T >::right
```

unique pointer to the right child

2.3.3.3 upper

```
template<class T>
node<T>* AP_node::node< T >::upper
```

pointer to the upper (parent) node. It cannot be unique as it can be possibly pointed by two different childern

2.3.3.4 value

```
template<class T>
T AP_node::node< T >::value
```

templated value contained by the node

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

- include/node.hpp

2.4 timer< Clock, Duration > Class Template Reference

```
#include <timer.hpp>
```

Public Member Functions

- void start ()
- double stop ()

2.4.1 Detailed Description

template<typename Clock = std::chrono::steady_clock, typename Duration = typename Clock::duration>
class timer< Clock, Duration >

class used in main_benchmark.cc in order to simply measure an initial and final times of an operation

2.4.2 Member Function Documentation

2.4.2.1 start()

```
template<typename Clock = std::chrono::steady_clock, typename Duration = typename Clock↩
::duration>
void timer< Clock, Duration >::start ( )  [inline]
```

set initial time

2.4.2.2 stop()

```
template<typename Clock = std::chrono::steady_clock, typename Duration = typename Clock↩
::duration>
double timer< Clock, Duration >::stop ( )  [inline]
```

set final time, diirectly returing the difference

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

- include/timer.hpp

Index