Binary Tree

1.0

Generated by Doxygen 1.9.0

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 BinaryTree< T > Class Template Reference

A generic Binary Tree class.

`#include <BinaryTree.hpp>`

Public Member Functions

- BinaryTree ()
  Default Constructor.
- BinaryTree (const BinaryTree< T > &copyTree)
  Copy Constructor.
- BinaryTree (BinaryTree< T > &&moveTree)
  Move Constructor.
- ~BinaryTree ()
  Class Destructor.
- void insert (const T element)
  Inserts element into the tree in level order, at first available position.
- bool bfsearch (const T &element)
  Breadth First Search.
- bool dfsearch (const T &element)
  Depth First Search.
- void remove (const T element)
  Removes the specified element from the tree.
- void clear ()
  Clears the entire tree recursively and resets all field elements to default.
- int size ()
  Returns the size of/number of nodes in the tree.
- bool empty ()
  Returns true if the tree is empty and false otherwise.
- int depth (const T &element)
  Returns depth of the specified element in tree.
- int height (const T &element)
  Returns height of the specified element in tree.

- void invertTree ()

  Recursively inverts the tree.
- void printInorder ()

  Prints tree Inorder.
- void printPreorder ()

  Prints tree Preorder.
- void printPostorder ()

  Prints tree Postorder.
- BinaryTree< T > & operator= (const BinaryTree &copyTree)

  Copy assignment operator.
- BinaryTree< T > & operator= (BinaryTree &&moveTree)

  Move assignment operator.

Private Member Functions

- int getDepth (const Node< T > *current, const T &element, int depth)

  Calculates the depth of an element in tree.
- int getHeight (const Node< T > *current, const T &element, int height, bool elementFound)

  Calculates the height of an element in tree.
- void invert (Node< T > *invertNode)

  Inverts all the left and right children of a node in the tree.
- void print (const Node< T > *node, const char &printOrder)

  Prints the binary tree.
- void swap (BinaryTree< T > &otherTree)

  Swaps Trees.

Private Attributes

- Node< T > * root
- int treeSize

Friends

- template<typename Type >
  std::ostream & **operator**<< (std::ostream &output, const BinaryTree< Type > &tree)

3.1.1 Detailed Description

template<typename T>
class BinaryTree< T >

A generic Binary Tree class.

This Binary Tree is templated to use any data type or class (typename T).

Template Parameters

T	Any data type or class.

Note

Any class or data type used with this tree class MUST implement its own Relational Operators (<, >, <=, >=, ==, !=) in order for this tree class to work correctly. All primitive data types (int, float, double, char, string, bool) already have this operator functionality, so no implementation for them is needed. But any custom class that is used with this tree class NEEDS to implement its own Relational Operators.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BinaryTree() [1/3]

```
template<typename T >
BinaryTree< T >::BinaryTree
```

Default Constructor.

Template Parameters

T	Any data type or class.

Initializes this tree object with a nullptr root and size of 0;

3.1.2.2 BinaryTree() [2/3]

```
template<typename T >
BinaryTree< T >::BinaryTree (
            const BinaryTree< T > & copyTree )
```

Copy Constructor.

Template Parameters

T	Any data type or class.

Parameters

copyTree	The tree whose contents will be copied into this tree object.

3.1.2.3 BinaryTree() [3/3]

```
template<typename T >
BinaryTree< T >::BinaryTree (
            BinaryTree< T > && moveTree )
```

Move Constructor.

Template Parameters

T	Any data type or class.

Parameters

moveTree	The tree whose contents will be moved into this tree object.

Note

std::move() needs to be used to call this constructor.

3.1.2.4 ~BinaryTree()

```
template<typename T >
BinaryTree< T >::~BinaryTree
```

Class Destructor.

Template Parameters

T	Any data type or class.

Recursively clears the tree using clear() function.

3.1.3 Member Function Documentation

3.1.3.1 bfsearch()

```
template<typename T >
bool BinaryTree< T >::bfsearch (
            const T & element )
```

Breadth First Search.

Template Parameters

T	Any data type or class.

Parameters

element	The element being searched for in the tree.

Returns

A boolean flag.

Returns true if the element is in the tree, otherwise returns false.

3.1.3.2 clear()

```
template<typename T >
void BinaryTree< T >::clear
```

Clears the entire tree recursively and resets all field elements to default.

Template Parameters

T	Any data type or class.

3.1.3.3 depth()

```
template<typename T >
int BinaryTree< T >::depth (
            const T & element )
```

Returns depth of the specified element in tree.

Template Parameters

T	Any data type or class.

Parameters

element	The element whose depth we are calculating.

Returns

Depth of element.

3.1.3.4 dfsearch()

```
template<typename T >
bool BinaryTree< T >::dfsearch (
            const T & element )
```

Depth First Search.

Template Parameters

T	Any data type or class.

Parameters

element	The element being searched for in the tree.

Returns

A boolean flag.

Returns true if the element is in the tree, otherwise returns false.

Note

This Depth First Search uses inorder traversal.

3.1.3.5 empty()

```
template<typename T >
bool BinaryTree< T >::empty
```

Returns true if the tree is empty and false otherwise.

Template Parameters

T	Any data type or class.

Returns

A boolean flag.

3.1.3.6 getDepth()

```
template<typename T >
int BinaryTree< T >::getDepth (
            const Node< T > * current,
            const T & element,
            int depth )  [private]
```

Calculates the depth of an element in tree.

Template Parameters

T	Any data type or class.

Parameters

current	The current Node we are looking at in tree.
element	The element whose depth we are calculating.
depth	The depth of the current node.

Returns

Depth of element.

3.1.3.7 getHeight()

```
template<typename T >
int BinaryTree< T >::getHeight (
            const Node< T > * current,
            const T & element,
            int height,
            bool elementFound )  [private]
```

Calculates the height of an element in tree.

Template Parameters

T	Any data type or class.

Parameters

current	The current Node we are looking at in tree.
element	The element whose height we are calculating.
height	The height of the current node.
elementFound	True if the element was found in the tree, otherwise false.

Returns

Height of element.

3.1.3.8 height()

```
template<typename T >
int BinaryTree< T >::height (
            const T & element )
```

Returns height of the specified element in tree.

Template Parameters

T	Any data type or class.

Parameters

element	The element whose height we are calculating.

Returns

Height of element.

3.1.3.9 insert()

```
template<typename T >
void BinaryTree< T >::insert (
            const T element )
```

Inserts element into the tree in level order, at first available position.

Template Parameters

T	Any data type or class.

Parameters

element	The element you want to add to the tree.

Breadth First insertion. No duplicates allowed.

3.1.3.10 invert()

```
template<typename T >
void BinaryTree< T >::invert (
            Node< T > * invertNode )  [private]
```

Inverts all the left and right children of a node in the tree.

Template Parameters

T	Any data type or class.

Parameters

invertNode	The node whose children will be inverted.

3.1.3.11 invertTree()

```
template<typename T >
void BinaryTree< T >::invertTree
```

Recursively inverts the tree.

Template Parameters

T	Any data type or class.

3.1.3.12 operator=() [1/2]

```
template<typename T >
BinaryTree< T > & BinaryTree< T >::operator= (
            BinaryTree< T > && moveTree )
```

Move assignment operator.

Template Parameters

T	Any data type or class.

Parameters

moveTree	The tree object from which to move elements.

Returns

A reference to a moved tree object.

Moves tree elements from the provided tree into this tree object. The provided tree object is empty after the move is complete.

Note

std::move() needs to be used to call this operator.

3.1.3.13 operator=() [2/2]

```
template<typename T >
BinaryTree< T > & BinaryTree< T >::operator= (
            const BinaryTree< T > & copyTree )
```

Copy assignment operator.

Template Parameters

T	Any data type or class.

Parameters

copyTree	The tree object from which to copy elements.

Returns

A reference to a copied tree object.

Copies a tree with the help of the copy constructor and a custom swap function.

3.1.3.14 print()

```
template<typename T >
void BinaryTree< T >::print (
            const Node< T > * node,
            const char & printOrder )  [private]
```

Prints the binary tree.

Template Parameters

T	Any data type or class.

Parameters

node	The root element in the tree.
printOrder	The order in which to print the tree (In-, Pre-, Post- order).

3.1.3.15 printInorder()

```
template<typename T >
void BinaryTree< T >::printInorder
```

Prints tree Inorder.

Template Parameters

T	Any data type or class.

3.1.3.16 printPostorder()

```
template<typename T >
void BinaryTree< T >::printPostorder
```

Prints tree Postorder.

Template Parameters

T	Any data type or class.

3.1.3.17 printPreorder()

```
template<typename T >
void BinaryTree< T >::printPreorder
```

Prints tree Preorder.

Template Parameters

T	Any data type or class.

3.1.3.18 remove()

```
template<typename T >
void BinaryTree< T >::remove (
            const T element )
```

Removes the specified element from the tree.

Template Parameters

T	Any data type or class.

Parameters

element	The element to be removed from the tree.

3.1.3.19 size()

```
template<typename T >
int BinaryTree< T >::size
```

Returns the size of/number of nodes in the tree.

Template Parameters

T	Any data type or class.

Returns

The size of the tree.

3.1.3.20 swap()

```
template<typename T >
void BinaryTree< T >::swap (
            BinaryTree< T > & other )  [private]
```

Swaps Trees.

Template Parameters

T	Any data type or class.

Parameters

other	The other tree with which to swap elements.

3.1.4 Member Data Documentation

3.1.4.1 root

```
template<typename T >
Node<T>* BinaryTree< T >::root  [private]
```

The root of the tree.

3.1.4.2 treeSize

```
template<typename T >
int BinaryTree< T >::treeSize  [private]
```

The size of the tree.

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

- BinaryTree.hpp

3.2 Node< T > Struct Template Reference

The Node struct is meant to hold the element and pointers to the left and right child elements.

`#include <BinaryTree.hpp>`

Public Member Functions

- Node (const T &element)
- Node (T &&element)
- ~Node ()

Public Attributes

- T element
- Node< T > * left
- Node< T > * right

Friends

- std::ostream & **operator**<< (std::ostream &output, const Node< T > &node)

3.2.1 Detailed Description

template<typename T>
struct Node< T >

The Node struct is meant to hold the element and pointers to the left and right child elements.

Template Parameters

T	Any data type or class.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Node() [1/2]

```
template<typename T >
Node< T >::Node (
            const T & element )  [inline]
```

Copy Constructor.

3.2.2.2 Node() [2/2]

```
template<typename T >
Node< T >::Node (
            T && element )  [inline]
```

Move Constructor.

3.2.2.3 ~Node()

```
template<typename T >
Node< T >::~Node ( )  [inline]
```

Struct Destructor.

3.2.3 Member Data Documentation

3.2.3.1 element

```
template<typename T >
T Node< T >::element
```

The element.

3.2.3.2 left

```
template<typename T >
Node<T>* Node< T >::left
```

Pointer to the left child node in the tree.

3.2.3.3 right

```
template<typename T >
Node<T>* Node< T >::right
```

Pointer to the right child node in the tree.

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

- BinaryTree.hpp

Chapter 4

File Documentation

4.1 BinaryTree.hpp File Reference

A generic Binary Tree data structure.

```
#include <queue>
#include <stack>
#include <iostream>
```

Classes

- struct Node< T >

  The Node struct is meant to hold the element and pointers to the left and right child elements.
- class BinaryTree< T >

  A generic Binary Tree class.

Functions

- template<typename T >
  std::ostream & operator<< (std::ostream &output, const BinaryTree< T > &tree)

  Output stream operator.

4.1.1 Detailed Description

A generic Binary Tree data structure.

Author

Al Timofeyev

Date

January 29, 2021

Version: 1.0 Modified By: Modified Date:

4.1.2 Function Documentation

4.1.2.1 operator<<()

```
template<typename T >
std::ostream& operator<< (
            std::ostream & output,
            const BinaryTree< T > & tree )
```

Output stream operator.

Template Parameters

T	Any data type or class.

Parameters

output	The output stream (usually std::cout).
tree	The tree object that will be printed.

Prints the tree elements to the specified output stream using inorder traversal.

Note

Any class or data type used with this tree class MUST implement its own operator<< in order for this operator<< to work correctly. All primitive data types (int, float, double, char, string, bool) already have this operator functionality so no implementation for them is needed. But any custom class that is used with this tree class NEEDS to implement its own operator<<.

Index