My Project

Generated by Doxygen 1.8.6

Tue Nov 29 2016 12:16:24

Contents

1 Class Index

1.1 Class List

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

2 File Index

2.1 File List

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

3 Class Documentation

3.1 BinaryNode Class Reference

Public Member Functions

- **BinaryNode** (int)
- **BinaryNode** (int, BinaryNode ∗, BinaryNode ∗)
- void **setItem** (int)
- int **getItem** () const
- bool **isLeaf** () const
- int **getNumOfChildren** () const
- BinaryNode ∗ **getLeftChildPtr** () const
- BinaryNode ∗ **getRightChildPtr** () const
- void **setLeftChildPtr** (BinaryNode ∗)
- void **setRightChildPtr** (BinaryNode ∗)

Private Attributes

- int **data**
- BinaryNode ∗ **leftChildPtr**
- BinaryNode ∗ **rightChildPtr**

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

- BinaryNode.h
- BinaryNode.cpp

3.2 BinarySearchTree Class Reference

Public Member Functions

- BinarySearchTree ()

  Constructor for class BinarySearchTree.
- BinarySearchTree (int)

  Constructor for class BinarySearchTree.
- ∼BinarySearchTree ()

  Destructor for class BinarySearchTree.
- int getHeightHelper (BinaryNode ∗) const

  Gets the height of the tree.
- int getNumberOfNodesHelper (BinaryNode ∗) const

  Gets the number of nodes in the tree.
- BinaryNode ∗ placeNode (BinaryNode ∗, BinaryNode ∗)

  Places a node in the correct spot.
- BinaryNode ∗ removeValue (BinaryNode ∗, int, bool &)

  Searches for the value that will be removed.
- BinaryNode ∗ removeNode (BinaryNode ∗)

  Removes a node from the tree.
- BinaryNode ∗ removeLeftmostNode (BinaryNode ∗, int &)

  Shifts the chain of nodes correctly to avoid disconnection.
- void destroyTree (BinaryNode ∗)

  Destroys the tree.
- void preorder (void visit(int &), BinaryNode ∗) const

  Traverses the tree using preorder.

- void inorder (void visit(int &), BinaryNode ∗) const

  Traverses the tree using inorder.
- void postorder (void visit(int &), BinaryNode ∗) const

  Traverses the tree using postorder.
- bool isEmpty () const

  Checks if tree is empty.
- int getHeight () const

  Calls recursive getHeightHelper function.
- int getNumberOfNodes () const

  Calls recursive getNumberOfNodesHelper function.
- int getRootData () const

  gets the data stored in rootPtr
- BinaryNode ∗ getRootNode () const

  Returns rootPtr.
- void setRootData (int)

  Sets the data stored in rootPtr to the parameter.
- bool add (int)

  Calls recursive placeNode function.
- bool remove (int)

  Calls recursive removeValue function.
- void clear ()

  Calls recursive destroyTree function.
- void preorderTraverse (void visit(int &)) const

  Calls recursive preorder function.
- void inorderTraverse (void visit(int &)) const

  Calls recursive inorder function.
- void postorderTraverse (void visit(int &)) const

  Calls recursive postorder function.

Private Attributes

- BinaryNode ∗ **rootPtr**

3.2.1 Constructor & Destructor Documentation

3.2.1.1 BinarySearchTree::BinarySearchTree ( )

Constructor for class BinarySearchTree.

Able to construct a BinarySearchTree object

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.1.2 BinarySearchTree::BinarySearchTree (int *a*)

Constructor for class BinarySearchTree.

Able to construct a BinarySearchTree object with given parameters

Precondition

None

Postcondition

None

None

Parameters

a	- rootPtr will contain this value as an item

Exceptions

None	

Note

: None

3.2.1.3 BinarySearchTree::~BinarySearchTree ( )

Destructor for class BinarySearchTree.

Able to destruct a BinarySearchTree object

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2 Member Function Documentation

3.2.2.1 bool BinarySearchTree::add (int *newData*)

Calls recursive placeNode function.

Calls placeNode function passing the rootPtr to it as a starting node

Precondition

None

Postcondition

None

Algorithm

If rootPtr is null, then allocates memory for it and creates the rootNode, else it allocates memory for a new node and calls placeNode to place it

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.2 void BinarySearchTree::clear ( )

Calls recursive destroyTree function.

Calls destroyTree function passing the rootPtr to it as a starting node

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.3 void BinarySearchTree::destroyTree (BinaryNode * *subTreePtr*)

Destroys the tree.

Recursively destroys the tree from the bottom up

Precondition

None

Postcondition

None

Algorithm

Recursively calls destroyTree until the bottom of it is reached, then begins to delete the nodes from the bottom up and set them to nullptr

Parameters

subTreePtr	- the node that will be deleted after the bottom-most nodes are

Exceptions

None	

Note

: None

3.2.2.4 int BinarySearchTree::getHeight ( ) const

Calls recursive getHeightHelper function.

Calls getHeightHelper function passing the rootPtr to it as a starting node

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.5 int BinarySearchTree::getHeightHelper (BinaryNode * *subTreePtr*) const

Gets the height of the tree.

Recursively gets the height of the tree

Precondition

None

Postcondition

None

Algorithm

Returns 0 if there is no node, returns a 1 + the max between a recursive call to the left side of the subtree and a recursive call to the right side of the subtree. This will eventually get you how many layers the tree has in total, which is the height of the tree

Parameters

subTreePtr	- used as the root of the subTree, will call its left side or right side depending on which one has more layers

Exceptions

None	

Note

: None

3.2.2.6 int BinarySearchTree::getNumberOfNodes ( ) const

Calls recursive getNumberOfNodesHelper function.

Calls getNumberOfNodesHelper function passing the rootPtr to it as a starting node

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.7 int BinarySearchTree::getNumberOfNodesHelper (BinaryNode * *subTreePtr*) const

Gets the number of nodes in the tree.

Recursively gets the number of nodes in the tree

Precondition

None

Postcondition

None

Algorithm

Returns 0 if there is no node, returns a 1 + a recursive call to the left side of the subtree + a recursive call to the right side of the subtree. This will eventually get you the total number of nodes

Parameters

subTreePtr	- used as the root of the subTree, will call its left side or right side depending on which one has more layers

Exceptions

None	

Note

: None

3.2.2.8 int BinarySearchTree::getRootData ( ) const

gets the data stored in rootPtr

returns the data being stored in rootPtr

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.9 BinaryNode * BinarySearchTree::getRootNode ( ) const

Returns rootPtr.

None

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.10 void BinarySearchTree::inorder (void *visitint &,* BinaryNode * *treePtr*) const

Traverses the tree using inorder.

Traverses the tree recursively, calling visit after it traverses the left side of the tree and before it traverses the right side

Precondition

None

Postcondition

None

Algorithm

Gets the item in treePtr and calls visit using that item in the correct order

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.11 void BinarySearchTree::inorderTraverse (void *visitint &*) const

Calls recursive inorder function.

Calls inorder function passing the rootPtr to it as a starting node

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.12 bool BinarySearchTree::isEmpty ( ) const

Checks if tree is empty.

Returns true if height is 0 and false if it's not

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.13 BinaryNode * BinarySearchTree::placeNode (BinaryNode * *subTreePtr,* BinaryNode * *newNodePtr*)

Places a node in the correct spot.

Recursively places a node in the correct numerical position

Precondition

None

Postcondition

None

Algorithm

If there is no node there, it returns the newNode. In add, it will add this newNode at the location where there is no node. If there is a node, then it will recursively call either the left side of the sub tree or the right side depending on the comparison of the values in each side with the value in the new node

Parameters

subTreePtr	- a pointer that points to the root of the subTree newNodePtr - a pointer that points to the node that contains the value that's being inserted

Exceptions

None	

Note

: None

3.2.2.14 void BinarySearchTree::postorder (void *visitint &,* BinaryNode * *treePtr*) const

Traverses the tree using postorder.

Traverses the tree recursively, calling visit after it traverses the rest of the tree

Precondition

None

Postcondition

None

Algorithm

Gets the item in treePtr and calls visit using that item in the correct order

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.15 void BinarySearchTree::postorderTraverse (void *visitint &*) const

Calls recursive postorder function.

Calls postorder function passing the rootPtr to it as a starting node

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.16 void BinarySearchTree::preorder (void *visitint &,* **BinaryNode** ∗ *treePtr*) const

Traverses the tree using preorder.

Traverses the tree recursively, calling visit before it traverses the rest of the tree

Precondition

None

Postcondition

None

Algorithm

Gets the item in treePtr and calls visit using that item in the correct order

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.17 void BinarySearchTree::preorderTraverse (void *visitint &*) const

Calls recursive preorder function.

Calls preorder function passing the rootPtr to it as a starting node

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.18 bool BinarySearchTree::remove (int *target*)

Calls recursive removeValue function.

Calls removeValue function passing the rootPtr to it as a starting node

Precondition

None

Postcondition

None

None

Parameters

None	

Exceptions

None	

Note

: None

3.2.2.19 BinaryNode * BinarySearchTree::removeLeftmostNode (BinaryNode * *nodePtr,* int & *inorderSuccessor*)

Shifts the chain of nodes correctly to avoid disconnection.

Moves the left side of the node being removed up the chain so the node can be removed correctly

Precondition

None

Postcondition

None

Algorithm

If there is no left child node, then inorderSuccessor gets the current item and removeNode is called, which should effectively handle one or no children without calling removeLeftMostNode again. If there is a left child node, then recrusively calls removeLeftMostNode. Once the bottom of the chain is reached, tempPtr gets the node that was removed. nodePtr sets tempPtr as itsleft child and then we begin to make our way back up by returning nodePtr from the various recursive function calls until the whole tree has shifted from that original subTreePtr that was originally being removed

Parameters

nodePtr	- the left subTreePtr of the original subTreePtr inorderSuccessor - an integer passed by reference so the tree can be shifted without losing any of its data

Exceptions

None	

Note

: None

3.2.2.20 BinaryNode * BinarySearchTree::removeNode (BinaryNode * *nodePtr*)

Removes a node from the tree.

Removes a node from the tree, calls removeLeftMostNode if the tree has 2 children

Precondition

None

Postcondition

None

Algorithm

Gets the number of children of the node, if there are no children, then the node gets deleted and set to null. If there is one child, then a node to connect is built. If the one child is on the right then the node to connect gets the right child and vice versa. The original node then gets deleted but not set to null because it is being replaced by the child node. If there are two children, then removeLeftMostNode is called. The right child is then set to what this function returns and the item is set to what this function changed it to be

Parameters

nodePtr	- points to the node that will be deleted

Exceptions

None	

Note

: None

3.2.2.21 BinaryNode * BinarySearchTree::removeValue (BinaryNode * *subTreePtr,* int *target,* bool & *isSuccessful*)

Searches for the value that will be removed.

Recursively searches for the value that will be removed

Precondition

None

Postcondition

None

Algorithm

Recursively calls itself until the value is found, and then calls removeNode once it is found, sets isSuccessful to false if the node does not exist

Parameters

subTreePtr	- a pointer that points to the root of the subTree target - an integer that is contained by the node that is targeted for removal isSuccessful - a bool that returns true if operation is successful

Exceptions

None	

Note

: None

3.2.2.22 void BinarySearchTree::setRootData (int *data*)

Sets the data stored in rootPtr to the parameter.

None

Precondition

None

Postcondition

None

None

Parameters

data	- rootPtr's item gets set to this integer

Exceptions

None	

Note

: None

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

- BinarySearchTree.h
- BinarySearchTree.cpp

4 File Documentation

4.1 BinarySearchTree.cpp File Reference

Implementation file for BinarySearchTree class.

```
#include "BinarySearchTree.h"
```

4.1.1 Detailed Description

Implementation file for BinarySearchTree class.

Author

Alex Kastanek

Implements all member methods of the BinarySearchTree class

Version

1.00 C.S. Student (15 November 2016) Initial development and testing of BinarySearchTree class

Note

Requires BinarySearchTree.h
None

4.2 BinarySearchTree.h File Reference

Definition file for BinarySearchTree class.

```
#include <iostream>
#include <fstream>
#include <cstddef>
#include "BinaryNode.h"
```

Classes

- class BinarySearchTree

4.2.1 Detailed Description

Definition file for BinarySearchTree class.

Author

Alex Kastanek

Specifies all member methods of the BinarySearchTree class

Version

1.00 C.S. Student (15 November 2016) Initial development and testing of BinarySearchTree class

Note

None

Index