Project 6

Generated by Doxygen 1.8.14

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

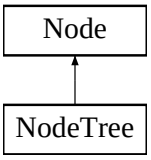
Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Class Documentation

3.1 Node Class Reference

Inheritance diagram for Node:



Public Member Functions

- Node ()

  Default Constructor for Node.
- Node (const int &anitem)
- Node (const int &item, Node ∗leftPtr, Node ∗rightPtr)

  Constructor for item, and the child objects.
- Node ∗ setItem (const int &anitem)

  Will set the value for the item.
- int getItem ()

  Will return the item value at that node.
- void **setCount** ()
- int **getCount** ()
- bool isLeaf () const

  Whill cheak if that node has leaft values.
- Node ∗ getLeftChildPtr ()

  Will return the left child pointer at that node.
- Node ∗ getRightChildPtr ()

  Will return the right child pointer at that node.
- void setLeftChildPtr (Node ∗leftPtr)

  Will set the left child pointer for the item.
- void setRightChildPtr (Node ∗rightPtr)

  Will set the right child pointer for the node.

3.1.1 Constructor & Destructor Documentation

3.1.1.1 Node() [1/2]

```
Node::Node (
            const int & anitem )
```

Constructor for the item only

Parameters

anitem	

Precondition

Will take in an int to give to the item

Postcondition

Will give the set the item to the value and set the childs to null

3.1.1.2 Node() [2/2]

```
Node::Node (
            const int & anitem,
            Node * leftPtr,
            Node * rightPtr )
```

Constructor for item, and the child objects.

Parameters

anitem	
leftPtr	
rightPtr	

Precondition

Will take the item and child pointer values

3.1.2 Member Function Documentation

3.1.2.1 getItem()

```
int Node::getItem ( )
```

Will return the item value at that node.

Returns

item

Precondition

Will get called at that node

Postcondition

Will return the value at that node

3.1.2.2 getLeftChildPtr()

```
Node * Node::getLeftChildPtr ( )
```

Will return the left child pointer at that node.

Returns

leftChildPtr

Precondition

Will get called at that node

Postcondition

Will return the left child pointer at that node

3.1.2.3 getRightChildPtr()

```
Node * Node::getRightChildPtr ( )
```

Will return the right child pointer at that node.

Returns

rightChildPtr

Precondition

Will get called at that node

Postcondition

Will return the right child pointer at that node

3.1.2.4 isLeaf()

```
bool Node::isLeaf ( ) const
```

Whill cheak if that node has leaft values.

Returns

bool

Precondition

Will check it that node has leafs

Postcondition

Will return a bool value to check the leafs

3.1.2.5 setItem()

```
Node * Node::setItem (
            const int & anitem )
```

Will set the value for the item.

Parameters

anitem	

Precondition

Will take in the item value to set it

Postcondition

Will set the item to the node item

3.1.2.6 setLeftChildPtr()

```
void Node::setLeftChildPtr (
            Node * leftPtr )
```

Will set the left child pointer for the item.

Parameters

leftPtr	

Precondition

Will take in the pointer value to set it

Postcondition

Will set the left child pointer to the node item

3.1.2.7 setRightChildPtr()

```
void Node::setRightChildPtr (
            Node * rightPtr )
```

Will set the right child pointer for the node.

Parameters

rightPtr	

Precondition

Will take in the pointer value to set it

Postcondition

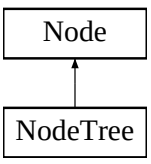
Will set the right child pointer to the node

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

- Node.h
- Node.cpp

3.2 NodeTree Class Reference

Inheritance diagram for NodeTree:



Public Member Functions

- NodeTree ()

  The Default constructor for the NodeTree.
- **NodeTree** (const int &rootItem)
- **NodeTree** (const NodeTree &tree)
- Node * insertInorder (Node *subTreePtr, Node *newNode)

  global variable to count the amount of nodes
- Node * getRootPtr ()
- virtual ~NodeTree ()
- bool isEmpty (Node *nodePtr) const

  Will test if the tree is empty.
- int getHeight ()

  Will get the height by calling the helper.
- int heightHelper (Node *treePtr)
- int getNumberofNodes (Node *nodePtr) const

  Will return the number of nodes in the tree. It will run until both children get null.
- int **getRootData** ()
- bool **remove** (const int &data)
- void setRootData (const int &newData)
- void setRoot (Node *root)
- bool add (const int &newData)
- void clear (Node *nodePtr)
- int getEntry (const int &anEntry)

  Will find the entry and return the value, if not it will return 0.
- bool contains (const int &anEntry) const
- void preorderTransverse (Node *nodePtr) const

  Will preorder transverse the tree and print the data.
- void inorderTransverse (Node *nodePtr) const

  Will inorder transverse the tree and print the data.
- void postorderTransverse (Node *nodePtr) const

  Will postorder transverse the tree and print the data.
- Node * deleteNode (Node *nodePtr, int key)

  Will delete the node with the key.
- Node * minValue (Node *nodePtr)
- void resetCount ()
- void inorderDelete (Node *nodePtr, int)

  Will delete the nodes that are equal to the nodes in BTS2.

Protected Member Functions

- Node * removeValue (Node *subTreePtr, int target, bool &success)
- Node * removeNode (Node *nodePtr)

  Will remove the node.
- Node * removeLeftmostNode (Node *subTreePtr, int &inorderSuccessor)
- Node * findNode (Node *treePtr, const int &target) const

  Will find the node with that value and return the address of it.

3.2.1 Constructor & Destructor Documentation

3.2.1.1 ~NodeTree()

```
NodeTree::~NodeTree ( )  [virtual]
```

Destructor and will call clear to delete the nodes

3.2.2 Member Function Documentation

3.2.2.1 add()

```
bool NodeTree::add (
            const int & newData )
```

This is the function used to call the insertInorder function

Parameters

newData	

Returns

true

Precondition

It will take the new data member to give to the insert

Postcondition

Will create a new node with the data member in it. Then it will pass that new node and the root ptr to add to the tree

3.2.2.2 clear()

```
void NodeTree::clear (
            Node * nodePtr )
```

Will recursively call itself to clear all the children and delete the root

Parameters

nodePtr	

Precondition

Will take in the root

Postcondition

Will clear all the children and nodes recursively

3.2.2.3 contains()

```
bool NodeTree::contains (
            const int & anEntry ) const
```

Will check if the tree contains that value

Parameters

anEntry	

Returns

bool

Precondition

Will take in a int value to find

Postcondition

Will call findnode and if that node contains the entry it will return true, else false

3.2.2.4 deleteNode()

```
Node * NodeTree::deleteNode (
            Node * nodePtr,
            int key )
```

Will delete the node with the key.

Parameters

nodePtr	
key	

Returns

nodePtr

Precondition

Will take in the root and key to delete that node

Postcondition

Will recursively call itself and set the children to the correct places after the deletion

3.2.2.5 findNode()

```
Node * NodeTree::findNode (
            Node * treePtr,
            const int & target ) const  [protected]
```

Will find the node with that value and return the address of it.

Parameters

treePtr	
target	

Returns

treePtr, will return the address to that target

Precondition

Will take in the root and the taret to find it

Postcondition

Will recursively find the target and return the address of the node

3.2.2.6 getEntry()

```
int NodeTree::getEntry (
            const int & anEntry )
```

Will find the entry and return the value, if not it will return 0.

Parameters

anEntry	

Returns

anEntry

Precondition

Will take in a int value to pass to findnode

Postcondition

Will call findnode to get the item, and if its the same it will return that value, if not it will return 0

3.2.2.7 getHeight()

```
int NodeTree::getHeight ( )
```

Will get the height by calling the helper.

Returns

The int value given by the helper

Precondition

Postcondition

Will chall the helper function to return the number

3.2.2.8 getNumberofNodes()

```
int NodeTree::getNumberofNodes (
            Node * nodePtr ) const
```

Will return the number of nodes in the tree. It will run until both children get null.

Parameters

nodePtr	

Returns

nodes

Precondition

Will take the rootptr

Postcondition

Will recursively call itself and increment the node counter to return the total value.

3.2.2.9 getRootPtr()

```
Node * NodeTree::getRootPtr ( )
```

Will return the rootPtr

Returns

rootPtr

3.2.2.10 heightHelper()

```
int NodeTree::heightHelper (
            Node * treePtr )
```

The heightHelper because it can access the rootPtr itself

Parameters

treePtr	

Returns

the max height

Precondition

Will take the rootptr to pass to the children

Postcondition

Will recusievly call itself until if finds the nullptr, it will increment itself when it calls itself and returns the max number

3.2.2.11 inorderDelete()

```
void NodeTree::inorderDelete (
            Node * nodePtr,
            int key )
```

Will delete the nodes that are equal to the nodes in BTS2.

Parameters

nodePtr	
key	

Precondition

Will take the root and the key

Postcondition

Will check if the key and the item in that node are equal, if equal it will call delete node to delete that node

3.2.2.12 inorderTransverse()

```
void NodeTree::inorderTransverse (
            Node * nodePtr ) const
```

Will inorder transverse the tree and print the data.

Parameters

nodePtr	

Precondition

Will get the rootptr to transverse the data

Postcondition

Will recursively call itself to move through the tree to print it in inorder

3.2.2.13 insertInorder()

```
Node * NodeTree::insertInorder (
            Node * subTreePtr,
            Node * newNode )
```

global variable to count the amount of nodes

Will insert the values into the tree

Parameters

subTreePtr,pointer	to the root
newNode,pointer	the a newNode that contains new value

Returns

A pointer to the root

Precondition

Will take in the root pointer and a new node containing the value of the new value

Postcondition

Will take the root pointer and the new node and set the new node to the right place

3.2.2.14 isEmpty()

```
bool NodeTree::isEmpty (
            Node * nodePtr ) const
```

Will test if the tree is empty.

Parameters

nodePtr	

Returns

bool, if it is empty

Precondition

Will take in the root pointer

Postcondition

Will recursively check if the node points to null, or both children are null

3.2.2.15 minValue()

```
Node * NodeTree::minValue (
            Node * nodePtr )
```

Will find the min value in the tree

Parameters

nodePtr	

Returns

current, pointer to the address of the min value

Precondition

Will take in the root to find the mon value

Postcondition

Will run down the left pointers until it finds null and return the address where it is min value

3.2.2.16 postorderTransverse()

```
void NodeTree::postorderTransverse (
            Node * nodePtr ) const
```

Will postorder transverse the tree and print the data.

Parameters

nodePtr	

Precondition

Will get the rootptr to transverse the data

Postcondition

Will recursively call itself to move through the tree to print it in postorder

3.2.2.17 preorderTransverse()

```
void NodeTree::preorderTransverse (
            Node * nodePtr ) const
```

Will preorder transverse the tree and print the data.

Parameters

nodePtr	

Precondition

Will get the rootptr to transverse the data

Postcondition

Will recursively call itself to move through the tree to print it in preorder

3.2.2.18 removeLeftmostNode()

```
Node * NodeTree::removeLeftmostNode (
            Node * subTreePtr,
            int & inorderSuccessor )  [protected]
```

Will remove the left most Node

Parameters

subTreePtr	
inorderSuccessor	

Returns

will wall the remove node to remove that node

Precondition

Will take in the root pointer and the value to set it

Postcondition

Will get traverse to get to the left most node and then call removeNode to delete it

3.2.2.19 removeNode()

```
Node * NodeTree::removeNode (
            Node * nodePtr )  [protected]
```

Will remove the node.

Parameters

nodePtr	

Returns

nodePtr

Precondition

Will take in the pointer to delete the node

Postcondition

Will take the pointer and delete that node and then fill in the gap

3.2.2.20 removeValue()

```
Node * NodeTree::removeValue (
            Node * subTreePtr,
            int target,
            bool & success )  [protected]
```

Will remove a value from the tree

Parameters

subTreePtr	
target	
success	

Returns

a pointer the the node

Precondition

Will take in the root , the target and the bool value

Postcondition

It will recursively call itself to remove the target and return that pointer

3.2.2.21 resetCount()

```
void NodeTree::resetCount ( )
```

Will reset the node value for getNumberofNodes

Postcondition

Will reset the global variable nodes to 1

3.2.2.22 setRoot()

```
void NodeTree::setRoot (
            Node * root )
```

Will rest the rootPtr to a address

Parameters

root	

Precondition

Will take in a node pointer

Postcondition

Will set the rootPtr to the root address

3.2.2.23 setRootData()

```
void NodeTree::setRootData (
            const int & newData )
```

Will set the s

Parameters

newData	

Precondition

Will take in a in to give to the root ptr

Postcondition

Will call the constructor for a new node and give the root pointer that data

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

- NodeTree.h
- NodeTree.cpp

Index