Project 06 - Binary Search Tree

1.00

Generated by Doxygen 1.8.6

Mon Nov 28 2016 23:35:36

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 ()
- BinaryNode (const int &anItem)
- BinaryNode (const int &anItem, BinaryNode ∗leftPtr, BinaryNode ∗rightPtr)
- void setItem (const int &anItem)
- int getItem () const
- bool isLeaf () const
- BinaryNode ∗ getLeftChildPtr () const
- BinaryNode ∗ getRightChildPtr () const
- void setLeftChildPtr (BinaryNode ∗leftPtr)
- void setRightChildPtr (BinaryNode ∗rightPtr)

Private Attributes

- int **item**
- BinaryNode ∗ **leftChildPtr**
- BinaryNode ∗ **rightChildPtr**

3.1.1 Constructor & Destructor Documentation

3.1.1.1 BinaryNode::BinaryNode ( )

Default constructor for binary node

3.1.1.2 BinaryNode::BinaryNode (const int & *anItem*)

Partially parameterized constructor for binary node

Parameters

anItem	Number to be placed inside the node

3.1.1.3 BinaryNode::BinaryNode (const int & *anItem,* BinaryNode * *leftPtr,* BinaryNode * *rightPtr*)

Fully parameterized constructor for binary node

Parameters

anItem	Number to be placed inside the node
leftPtr	The node that will be placed on the left side of this node
rightPtr	The node that will be placed on the right side of this node

3.1.2 Member Function Documentation

3.1.2.1 int BinaryNode::getItem ( ) const

Returns the number at the node

return item stored in the node

3.1.2.2 BinaryNode * BinaryNode::getLeftChildPtr ( ) const

Find the node's left child

Returns

a pointer to the node's left child

3.1.2.3 BinaryNode * BinaryNode::getRightChildPtr ( ) const

Find the node's right child

Returns

a pointer to the node's right child

3.1.2.4 bool BinaryNode::isLeaf ( ) const

Returns if the node has no children

Returns

if the node doesn't have children

3.1.2.5 void BinaryNode::setItem (const int & *anItem*)

Changes integer item to the specified integer in parameter

Parameters

anItem	Number to be changed to

3.1.2.6 void BinaryNode::setLeftChildPtr (BinaryNode * *leftPtr*)

Set the node's left child

3.1.2.7 void BinaryNode::setRightChildPtr (BinaryNode * *rightPtr*)

Set the node's right child

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

- BinaryNode.h
- BinaryNode.cpp

3.2 BinarySearchTree Class Reference

Public Member Functions

- BinarySearchTree ()
- BinarySearchTree (const int &rootItem)
- ~BinarySearchTree ()
- bool isEmpty () const
- int getHeight () const
- int getNumberOfNodes () const
- bool add (const int &newData)
- bool remove (const int &data)
- void clear ()
- void preorderTraverse (void visit(int &)) const
- void inorderTraverse (void visit(int &)) const
- void postorderTraverse (void visit(int &)) const

Protected Member Functions

- int getHeightHelper (BinaryNode *subTreePtr) const
- int getNumberOfNodesHelper (BinaryNode *subTreePtr) const
- void destroyTree (BinaryNode *subTreePtr)
- BinaryNode * insertInorder (BinaryNode *subTreePtr, BinaryNode *newNode)
- BinaryNode * removeValue (BinaryNode *subTreePtr, const int target, bool &success)
- BinaryNode * removeNode (BinaryNode *nodePtr)
- BinaryNode * removeLeftmostNode (BinaryNode *subTreePtr, int &inorderSuccessor)
- void preorder (void visit(int &), BinaryNode *treePtr) const
- void inorder (void visit(int &), BinaryNode *treePtr) const
- void postorder (void visit(int &), BinaryNode *treePtr) const

Private Attributes

- BinaryNode * **rootPtr**

3.2.1 Constructor & Destructor Documentation

3.2.1.1 BinarySearchTree::BinarySearchTree ( )

Default constructor for binary search tree

3.2.1.2 BinarySearchTree::BinarySearchTree (const int & *rootItem*)

Fully parameterized constructor for binary search tree

Parameters

rootItem	the first item to be added to the tree

3.2.1.3 BinarySearchTree::~BinarySearchTree ( )

Deconstructor

Postcondition

Entire tree is deleted

3.2.2 Member Function Documentation

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

Interface for adding a new data element into the tree

Postcondition

specified data is entered into the tree

Parameters

newData	data to be put into the tree

Returns

if the insertion was successful (always true here)

3.2.2.2 void BinarySearchTree::clear ( )

Removes everything in the tree

Postcondition

entirety of tree's contents have been removed

3.2.2.3 void BinarySearchTree::destroyTree (BinaryNode * *subTreePtr*) [protected]

Sends the (sub)tree to oblivion

Postcondition

everything under the under is deleted

Parameters

subTreePtr	Node to start deleting from

3.2.2.4 int BinarySearchTree::getHeight ( ) const

Interface for finding the height

Returns

the height of the tree found by the called upon recursive function

3.2.2.5 int BinarySearchTree::getHeightHelper (BinaryNode * *subTreePtr*) const [protected]

Find the height of the tree recursively

Parameters

subTreePtr	Node to start counting from

Returns

The height of the tree

3.2.2.6 int BinarySearchTree::getNumberOfNodes ( ) const

Interface for finding the amount of nodes

Returns

the number of nodes found by the called upon recursive function

3.2.2.7 int BinarySearchTree::getNumberOfNodesHelper (BinaryNode * *subTreePtr*) const [protected]

Finds the number of nodes recursively

Parameters

subTreePtr	Node to start counting from

Returns

The number of nodes in the tree

3.2.2.8 void BinarySearchTree::inorder (void *visitint &,* BinaryNode * *treePtr*) const [protected]

Visits the nodes in order by value, starting from specified node

Parameters

visit	Function to do things when visiting the node
treePtr	Node to start visitng from

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

Interface for visiting the tree in order

Parameters

visit	Function to do things when visiting the node

3.2.2.10 BinaryNode * BinarySearchTree::insertInorder (BinaryNode * *subTreePtr,* BinaryNode * *newNodePtr*) [protected]

Inserts a new node into the tree in order

Postcondition

node is inserted into the tree based on value

Parameters

subTreePtr	the current node to compare value with
newNodePtr	the new node to be inserted

Returns

the node where the new node was inserted at

3.2.2.11 bool BinarySearchTree::isEmpty ( ) const

Check whether the tree is empty or not

Returns

whether rootPtr points to null (empty)

3.2.2.12 void BinarySearchTree::postorder (void *visitint &,* **BinaryNode** ∗ *treePtr*) const [protected]

Visits the nodes in post order, starting from specified node

Parameters

visit	Function to do things when visiting the node
treePtr	Node to start visitng from

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

Interface for visiting the tree in pstorder

Parameters

visit	Function to do things when visiting the node

3.2.2.14 void BinarySearchTree::preorder (void *visitint &,* **BinaryNode** ∗ *treePtr*) const [protected]

Visits the nodes in preorder, starting from specified node

Parameters

visit	Function to do things when visiting the node
treePtr	Node to start visitng from

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

Interface for visiting the tree in preorder

Parameters

visit	Function to do things when visiting the node

3.2.2.16 bool BinarySearchTree::remove (const int & *data*)

Interface for removing a number from the tree

Parameters

data	integer to be removed from the tree

Returns

a bool signalling whether the operation was successful

3.2.2.17 BinaryNode * BinarySearchTree::removeLeftmostNode (BinaryNode * *subTreePtr,* int & *inorderSuccessor*) [protected]

Removes the smalles value of the (sub)tree

Parameters

subTreePtr	current node to start searching from
inorder-Successor	integer to store the value of the node

Returns

the node that was removed at the leftmost side of the tree

3.2.2.18 BinaryNode * BinarySearchTree::removeNode (BinaryNode * *nodePtr*) [protected]

Remove a node

Postcondition

the specified node is removed, and the rest of the tree is shifted accordingly

Parameters

nodePtr	the node to be removed

Returns

a pointer to the new node in the position

3.2.2.19 BinaryNode * BinarySearchTree::removeValue (BinaryNode * *subTreePtr*, const int *target*, bool & *success*) [protected]

Remove a certain value from the tree

Parameters

subTreePtr	current node to start searching from
target	the number to be removed
success	if the operation has succeeded

Returns

a null pointer if no value was removed, or the pointer to the node that replaced the old node's position after correctly shifting the tree

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

- BinarySearchTree.h
- BinarySearchTree.cpp

4 File Documentation

4.1 BinaryNode.cpp File Reference

Implementation file for binary node.

```
#include "BinaryNode.h"
```

4.1.1 Detailed Description

Implementation file for binary node. Some of the functions that are irrelevant to this assignment have been removed (and thus not implemented)

Version

1.00 Wei Tong (11 November 2016) Turned in version and initial development

4.2 BinaryNode.h File Reference

Header file for binary node.

Classes

- class BinaryNode

4.2.1 Detailed Description

Header file for binary node. Some of the functions that are irrelevant to this assignment have been removed (and thus not implemented)

Version

1.00 Wei Tong (11 November 2016) Turned in version and initial development

4.3 BinarySearchTree.cpp File Reference

Implementation file for binary node.

```
#include "BinarySearchTree.h"
#include <algorithm>
```

4.3.1 Detailed Description

Implementation file for binary node. Some of the functions that are irrelevant to this assignment have been removed (and thus not implemented)

Version

1.00 Wei Tong (11 November 2016) Turned in version and initial development

4.4 BinarySearchTree.h File Reference

Header file for binary search tree.

```
#include "BinaryNode.h"
```

Classes

- class BinarySearchTree

4.4.1 Detailed Description

Header file for binary search tree. Some of the functions that are irrelevant to this assignment have been removed (and thus not implemented)

Version

1.00 Wei Tong (11 November 2016) Turned in version and initial development

4.5 main.cpp File Reference

Test driver of binary search tree implementation.

```
#include "BinaryNode.h"
#include "BinarySearchTree.h"
#include <iostream>
#include <fstream>
#include <ctime>
```

Macros

- #define **SIZE_FIRST** 100
- #define **SIZE_SECOND** 10

Functions

- void removeTraverse (int &data)
- void traverseOutput (int &data)
- int **main** ()

Variables

- ofstream **fout**
- BinarySearchTree **first**
- BinarySearchTree **second**

4.5.1 Detailed Description

Test driver of binary search tree implementation. Main file for project. Will generate an amount of random data specified by SIZE_FIRST, between 1 and 200 and put them into the first binary search tree. A second set of random data, the amount specified by SIZE_SECOND will be created. This set however, will take a random amount of numbers between 1 and its max size, and fill that amount of number with random numbers from the first BST. The remaining numbers will be randomly generated. For both BSTs, there will be no repeat numbers.

Version

1.00 Wei Tong (11 November 2016) Turned in version and initial development

Note

Adapted from Frank M. Carrano and Timothy M. Henry Copyright (c) 2012 Pearson Education, Hoboken, New Jersey.

4.5.2 Function Documentation

4.5.2.1 void removeTraverse (int & *data*)

Removes data in the BST as it traverses

Postcondition

The first BST will have data removed from it as the BST that calls upon this traverse is traversed. All numbers that are in both the BSTs are removed from the first BST

Parameters

data	The number at the node currently being visited and will be removed from the first BST if it exists

4.5.2.2 void traverseOutput (int & *data*)

Removes data in the BST as it traverses

Postcondition

The data of the node currently being visited will be output to the file

Parameters

data	The number at the node currently being visited and will be outputted from the BST

Index