CS 302 Project 6

Generated by Doxygen 1.8.6

Mon Apr 13 2015 13:37:37

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

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 BinaryNode< ItemType > Class Template Reference

`#include <binarySearchTree.h>`

Public Member Functions

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

4.1.1 Detailed Description

template<class ItemType>class BinaryNode< ItemType >

Binary node class. Used in binary tree and binary search tree operations. Adapted from textbook code.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 template<class ItemType > BinaryNode< ItemType >::BinaryNode ( )

Binary node default constructor. Sets left and right pointers to null and does not assign data.

Precondition

None.

Postcondition

Object created with left and right pointers set to null.

Returns

None.

4.1.2.2 template< class ItemType > BinaryNode< ItemType >::BinaryNode (const ItemType & *anItem*)

Binary node parameterized constructor. Sets left and right pointers to null and assigns data to the node.

Precondition

None.

Postcondition

Object created with specified data, left and right pointers set to null.

Parameters

anItem	The data the node should contain.

Returns

None.

4.1.2.3 template< class ItemType > BinaryNode< ItemType >::BinaryNode (const ItemType & *anItem,* BinaryNode< ItemType > ∗ *leftPtr,* BinaryNode< ItemType > ∗ *rightPtr*)

Binary node parameterized constructor. Sets left and right pointers to specified trees and assigns data to the node.

Precondition

None.

Postcondition

Object created with specified data, left and right pointers pointing to specified subtrees.

Parameters

anItem	The data the node should contain.
leftPtr	The subtree the node's left pointer should point to.
rightPtr	The subtree the node's right pointer should point to.

Returns

None.

4.1.3 Member Function Documentation

4.1.3.1 template< class ItemType > ItemType BinaryNode< ItemType >::getItem ( ) const

Binary node getItem function. Returns the data stored in the node's item data member. Caveat emptor when calling this function with a node that has not yet had data assigned to it, likely to return garbage.

Precondition

None, but caveat emptor when calling on a node where data has not been assigned.

Postcondition

Data stored in item data member of the node returned. Node contents unchanged.

Returns

ItemType The object stored in the node's item data member.

4.1.3.2 template$<$class ItemType $>$ BinaryNode$<$ ItemType $>$ $*$ BinaryNode$<$ ItemType $>$::getLeftChildPtr ( ) const

Binary node getLeftChildPtr function.

Precondition

None.

Postcondition

leftChildPtr of the node returned; node contents unchanged

Returns

BinaryNode$<$ItemType$>*$ The left child pointer.

4.1.3.3 template$<$class ItemType $>$ BinaryNode$<$ ItemType $>$ $*$ BinaryNode$<$ ItemType $>$::getRightChildPtr ( ) const

Binary node getRightChildPtr function.

Precondition

None.

Postcondition

rightChildPtr of the node returned; node contents unchanged

Returns

BinaryNode$<$ItemType$>*$ The right child pointer.

4.1.3.4 template$<$class ItemType $>$ bool BinaryNode$<$ ItemType $>$::isLeaf ( ) const

Binary node isLeaf function. Checks to see if the node is a leaf (ie if both left and right pointers point to null, ie no children).

Precondition

None.

Postcondition

Returns true if the node is a leaf and false if not; node contents unchanged.

Returns

bool Whether the node is a leaf or not.

4.1.3.5 template<class ItemType > void BinaryNode< ItemType >::setItem (const ItemType & *anItem*)

Binary node setItem function. Sets the data stored by the node to the specified value.

Precondition

None.

Postcondition

Item data member holds anItem.

Parameters

anItem	The data the node should contain.

Returns

None.

4.1.3.6 template<class ItemType > void BinaryNode< ItemType >::setLeftChildPtr (BinaryNode< ItemType > ∗ *leftPtr*)

Binary node setLeftChildPtr function.

Precondition

None.

Postcondition

leftChildPtr of the node set to specified value.

Parameters

leftPtr	The pointer the left pointer value of the node should store.

Returns

None.

4.1.3.7 template<class ItemType > void BinaryNode< ItemType >::setRightChildPtr (BinaryNode< ItemType > ∗ *rightPtr*)

Binary node setRightChildPtr function.

Precondition

None.

Postcondition

rightChildPtr of the node set to specified value.

Parameters

rightPtr	The pointer the right pointer value of the node should store.

Returns

None.

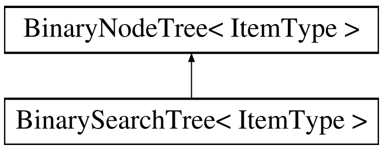
The documentation for this class was generated from the following files:

- binarySearchTree.h
- binarySearchTree.cpp

4.2 BinaryNodeTree< ItemType > Class Template Reference

```
#include <binarySearchTree.h>
```

Inheritance diagram for BinaryNodeTree< ItemType >:



Public Member Functions

- BinaryNodeTree ()
- BinaryNodeTree (const ItemType &rootItem)
- BinaryNodeTree (const ItemType &rootItem, const BinaryNodeTree< ItemType > ∗leftTreePtr, const Binary-NodeTree< ItemType > ∗rightTreePtr)
- BinaryNodeTree (const BinaryNodeTree< ItemType > &tree)
- virtual ∼BinaryNodeTree ()
- bool isEmpty () const
- int getHeight () const
- int getNumberOfNodes () const
- ItemType getRootData () const
- void setRootData (const ItemType &newData)
- bool add (const ItemType &newData)
- bool remove (const ItemType &data)
- void clear ()
- ItemType getEntry (const ItemType &anEntry) const
- bool contains (const ItemType &anEntry) const
- void preorderTraverse (void visit(ItemType &)) const
- void inorderTraverse (void visit(ItemType &)) const
- void postorderTraverse (void visit(ItemType &)) const
- BinaryNodeTree< ItemType > & operator= (const BinaryNodeTree &rightHandSide)

Protected Member Functions

- int getHeightHelper (BinaryNode< ItemType > ∗subTreePtr) const
- int getNumberOfNodesHelper (BinaryNode< ItemType > ∗subTreePtr) const
- void destroyTree (BinaryNode< ItemType > ∗subTreePtr)

- BinaryNode< ItemType > * balancedAdd (BinaryNode< ItemType > *subTreePtr, BinaryNode< ItemType > *newNodePtr)
- BinaryNode< ItemType > * removeValue (BinaryNode< ItemType > *subTreePtr, const ItemType target, bool &success)
- BinaryNode< ItemType > * moveValuesUpTree (BinaryNode< ItemType > *subTreePtr)
- BinaryNode< ItemType > * findNode (BinaryNode< ItemType > *treePtr, const ItemType &target, bool &success) const
- BinaryNode< ItemType > * copyTree (const BinaryNode< ItemType > *treePtr) const
- void preorder (void visit(ItemType &), BinaryNode< ItemType > *treePtr) const
- void inorder (void visit(ItemType &), BinaryNode< ItemType > *treePtr) const
- void postorder (void visit(ItemType &), BinaryNode< ItemType > *treePtr) const

4.2.1 Detailed Description

template<class ItemType>class BinaryNodeTree< ItemType >

Unsorted binary tree class. Uses binary nodes to build the tree. Adapted from textbook code.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 template<class ItemType > BinaryNodeTree< ItemType >::BinaryNodeTree ( )

Binary tree default constructor. Tree contains no nodes, so points to null.

Precondition

None.

Postcondition

Object created with root pointing to null.

Returns

None.

4.2.2.2 template<class ItemType > BinaryNodeTree< ItemType >::BinaryNodeTree (const ItemType & *rootItem*)

Binary tree parameterized constructor. Creates root node with specified data.

Precondition

None.

Postcondition

Object created with a root node containing rootItem.

Parameters

rootItem	The data the root node is to contain.

Returns

None.

4.2.2.3 template<class ItemType > BinaryNodeTree< ItemType >::BinaryNodeTree (const ItemType & *rootItem*, const BinaryNodeTree< ItemType > ∗ *leftTreePtr*, const BinaryNodeTree< ItemType > ∗ *rightTreePtr*)

Binary tree parameterized constructor. Creates root node with specified data, with subtrees that are copies of the left and right subtrees provided.

Precondition

None.

Postcondition

Object created with a root node containing rootItem and child pointers that point to copies of the subtrees provided.

Parameters

rootItem	The data the root node is to contain.
leftTreePtr	The subtree that is to be copied for the tree's left child.
rightTreePtr	The subtree that is to be copied for the tree's right child.

Returns

None.

4.2.2.4 template<class ItemType > BinaryNodeTree< ItemType >::BinaryNodeTree (const BinaryNodeTree< ItemType > & *tree*)

Binary tree copy constructor. Creates a new tree which is a copy of the tree provided.

Precondition

None.

Postcondition

Binary tree created which is a copy of tree provided.

Parameters

tree	The tree that this object is to copy.

Returns

None.

4.2.2.5 template< class ItemType > BinaryNodeTree< ItemType >::∼BinaryNodeTree ( ) [virtual]

Binary tree destructor. Deallocates any dynamic memory for nodes in this tree using destroyTree function.

Precondition

None.

Postcondition

All dynamic memory for nodes in this tree deallocated.

Returns

None.

4.2.3 Member Function Documentation

4.2.3.1 template< class ItemType > bool BinaryNodeTree< ItemType >::add (const ItemType & *newData*)

Binary tree add function. Adds specified value to the binary tree as a new node. Uses balancedAdd helper function to try to maintain balance in the tree after the add.

Precondition

None.

Postcondition

Value added to the tree; balance maintained.

Returns

bool Returns true, signifying successful add.

4.2.3.2 template< class ItemType > BinaryNode< ItemType > ∗ BinaryNodeTree< ItemType >::balancedAdd (BinaryNode< ItemType > ∗ *subTreePtr,* BinaryNode< ItemType > ∗ *newNodePtr*) [protected]

Binary tree balancedAdd function. Called by public add function. Adds a value to the binary tree in a way that maintains the balance of the tree. This tree is not sorted, so the value will not be placed in any sorted order. Uses recursion to find the proper place to add the node.

Precondition

None.

Postcondition

Value added to tree while maintaining the tree's balance.

Parameters

subTreePtr	The tree to which the value should be added.
newNodePtr	The node to be added to the tree.

Returns

BinaryNode<ItemType>∗ A pointer to the tree once the new value has been added.

4.2.3.3 template<class ItemType > void BinaryNodeTree< ItemType >::clear ( )

Binary tree clear function. Deallocates all nodes in the tree, discarding all data, and restores tree to an empty state. Uses destroyTree helper function.

Precondition

None.

Postcondition

All nodes deallocated and all data lost; tree is returned to an empty state.

Returns

None.

4.2.3.4 template<class ItemType > bool BinaryNodeTree< ItemType >::contains (const ItemType & *anEntry*) const

Binary tree contains function. Searches binary tree for target value. Returns true if value is found, false if not. Uses findNode helper function.

Precondition

None.

Postcondition

Tree contents unchanged; bool signifying whether the value was found returned.

Parameters

anEntry	The value to be found in the binary tree.

Returns

bool True if the value is in the tree, false if not.

4.2.3.5 template<class ItemType > BinaryNode< ItemType > ∗ BinaryNodeTree< ItemType >::copyTree (const BinaryNode< ItemType > ∗ *treePtr*) const [protected]

Binary tree copyTree function. Called by copy constructor and = operator. Uses recursion to visit the entire target tree.

Precondition

None.

Postcondition

Tree contents unchanged; tree contents copied to new tree.

Parameters

treePtr	The tree to be copied.

Returns

BinaryNode<ItemType>∗ A pointer to the new tree that has been created.

4.2.3.6 template<class ItemType > void **BinaryNodeTree**< ItemType >::destroyTree (**BinaryNode**< ItemType > ∗ *subTreePtr*) `[protected]`

Binary tree destroy function. Called by clear function and destructor to deallocate all dynamic memory for nodes in the tree. Uses recursion to visit all nodes.

Precondition

None.

Postcondition

All nodes of the tree pointed to by subTreePtr deallocated and data lost.

Parameters

subTreePtr	the subtree whose nodes are to be destroyed.

Returns

None.

4.2.3.7 template<class ItemType > **BinaryNode**< ItemType > ∗ **BinaryNodeTree**< ItemType >::findNode (**BinaryNode**< ItemType > ∗ *treePtr,* const ItemType & *target,* bool & *success*) const `[protected]`

Binary tree findNode function. Called by public getEntry and contains functions. Searches the tree for a node containing target value. Uses recursion to search the entire tree.

Precondition

None.

Postcondition

Tree contents unchanged; success variable updated to true if node found, pointer to the node containing the value returned if found.

Parameters

subTreePtr	The subtree to search for the value.
target	The target value that is to be found.
success	Bool denoting whether the value was found or not; updated to true if found.

Returns

BinaryNode<ItemType>∗ A pointer to the node containing the value.

4.2.3.8 template<class ItemType > ItemType BinaryNodeTree< ItemType >::getEntry (const ItemType & *anEntry*) const

Binary tree getEntry function. Searches binary tree for target value. Returns the value if found, otherwise prints an error message to console. Uses findNode helper function.

Precondition

None.

Postcondition

Tree contents unchanged; value returned if found, otherwise error written to console.

Parameters

anEntry	The value to be found in the binary tree.

Returns

ItemType The value found in the binary tree, if the search was successful.

4.2.3.9 template<class ItemType > int BinaryNodeTree< ItemType >::getHeight ( ) const

Binary tree getHeight function. Uses getHeightHelper function to determine the height of the tree.

Precondition

None.

Postcondition

Tree contents unchanged; height returned.

Returns

int The height of the binary tree (0 if empty).

4.2.3.10 template<class ItemType > int BinaryNodeTree< ItemType >::getHeightHelper (BinaryNode< ItemType > ∗ *subTreePtr*) const [protected]

Binary tree getHeightHelper function. Called by getHeight to determine the height of the tree. Uses recursion to find total height.

Precondition

None.

Postcondition

Height of subTreePtr returned; tree contents unchanged.

Parameters

subTreePtr	the subtree whose height is to be found.

Returns

int The height of the tree.

4.2.3.11 template< class ItemType > int BinaryNodeTree< ItemType >::getNumberOfNodes ( ) const

Binary tree getNumberOfNodes function. Uses getNumberOfNodesHelper function to determine the number of nodes in the tree.

Precondition

None.

Postcondition

Tree contents unchanged; number of nodes returned.

Returns

int The number of nodes in the binary tree (0 if empty).

4.2.3.12 template< class ItemType > int BinaryNodeTree< ItemType >::getNumberOfNodesHelper (BinaryNode< ItemType > * *subTreePtr*) const [protected]

Binary tree getNumberOfNodesHelper function. Called by getNumberOfNodes to determine the number of nodes in the tree. Uses recursion to find total number of nodes.

Precondition

None.

Postcondition

Number of nodes in subTreePtr returned; tree contents unchanged.

Parameters

subTreePtr	the subtree whose number of nodes is to be found.

Returns

int The number of nodes in the tree.

4.2.3.13 template< class ItemType > ItemType BinaryNodeTree< ItemType >::getRootData ( ) const

Binary tree getRootData function. Binary tree must contain some data. Returns item stored in the root node of the tree. Caveat emptor using this function on an empty binary tree (!!!).

Precondition

The binary tree contains at least one value.

Postcondition

Tree contents unchanged; value stored in root node returned.

Returns

ItemType The item stored in the root node.

4.2.3.14 template<class ItemType > void BinaryNodeTree< ItemType >::inorder (void *visitItemType &,* BinaryNode< ItemType > ∗ *treePtr*) const [protected]

Binary tree inorder traversal function. Visits left, then root, then right, performing an action specified in the user provided visit function. Uses recursion to visit the entire tree.

Precondition

None.

Postcondition

Tree contents unchanged; visit performed for each value in the tree in specified order.

Parameters

visit	The user provided visit function, which will perform some action at each node.
treePtr	The tree to be traversed.

Returns

None.

4.2.3.15 template<class ItemType > void BinaryNodeTree< ItemType >::inorderTraverse (void *visitItemType &*) const

Binary tree inorder traversal function. Visits left, then root, then right, performing an action specified in the user provided visit function. Uses preorder helper function.

Precondition

None.

Postcondition

Tree contents unchanged; visit performed for each value in the tree in specified order.

Parameters

visit	The user provided visit function, which will perform some action at each node.

Returns

None.

4.2.3.16 template< class ItemType > bool **BinaryNodeTree**< ItemType >::isEmpty ( ) const

Binary tree isEmpty function. Checks to see if there are any nodes in the binary tree.

Precondition

None.

Postcondition

Tree contents unchanged; bool returned.

Returns

bool True if the tree is empty, false if it is not empty.

4.2.3.17 template< class ItemType > **BinaryNode**< ItemType > ∗ **BinaryNodeTree**< ItemType >::moveValuesUpTree (**BinaryNode**< ItemType > ∗ *subTreePtr*) [protected]

Binary tree moveValuesUpTree function. Shifts nodes after the removal of a value to maintain the integrity of the pointers in the tree and not lose data. Uses recursion to loop through subtree values as necessary. Since this tree is not sorted, values are moved without any element of comparison.

Precondition

Called by removeValue function for target node that is to be removed.

Postcondition

After all recursion, the tree will be 'fixed' to account for the removed node.

Parameters

subTreePtr	Pointer to the node that is to be removed.

Returns

BinaryNode<ItemType>∗ A pointer to the tree once the operation has been carried out.

4.2.3.18 template< class ItemType > **BinaryNodeTree**< ItemType > & **BinaryNodeTree**< ItemType >::operator= (const **BinaryNodeTree**< ItemType > & *rightHandSide*)

Binary tree overloaded = operator. Sets left hand tree to be a copy of right hand tree using copyTree helper function.

Precondition

None.

Postcondition

Left hand tree is now a copy of right hand tree. Any nodes contained in the left hand tree before the copy deallocated and data lost.

Parameters

rightHandSide	The tree that is to be copied from.

Returns

BinaryNodeTree<ItemType> The tree that now contains the copy of the right hand side tree.

4.2.3.19 template<class ItemType > void BinaryNodeTree< ItemType >::postorder (void *visitItemType &*, BinaryNode< ItemType > ∗ *treePtr*) const `[protected]`

Binary tree postorder traversal function. Visits left, then right, then root, performing an action specified in the user provided visit function. Uses recursion to visit the entire tree.

Precondition

None.

Postcondition

Tree contents unchanged; visit performed for each value in the tree in specified order.

Parameters

visit	The user provided visit function, which will perform some action at each node.
treePtr	The tree to be traversed.

Returns

None.

4.2.3.20 template<class ItemType > void BinaryNodeTree< ItemType >::postorderTraverse (void *visitItemType &*) const

Binary tree postorder traversal function. Visits left, then right, then root, performing an action specified in the user provided visit function. Uses postorder helper function.

Precondition

None.

Postcondition

Tree contents unchanged; visit performed for each value in the tree in specified order.

Parameters

visit	The user provided visit function, which will perform some action at each node.

Returns

None.

4.2.3.21 template< class ItemType > void BinaryNodeTree< ItemType >::preorder (void *visitItemType &,* BinaryNode< ItemType > * *treePtr*) const [protected]

Binary tree preorder traversal function. Visits root, then left, then right, performing an action specified in the user provided visit function. Uses recursion to visit the entire tree.

Precondition

None.

Postcondition

Tree contents unchanged; visit performed for each value in the tree in specified order.

Parameters

visit	The user provided visit function, which will perform some action at each node.
treePtr	The tree to be traversed.

Returns

None.

4.2.3.22 template< class ItemType > void BinaryNodeTree< ItemType >::preorderTraverse (void *visitItemType &*) const

Binary tree preorder traversal function. Visits root, then left, then right, performing an action specified in the user provided visit function. Uses preorder helper function.

Precondition

None.

Postcondition

Tree contents unchanged; visit performed for each value in the tree in specified order.

Parameters

visit	The user provided visit function, which will perform some action at each node.

Returns

None.

4.2.3.23 template< class ItemType > bool BinaryNodeTree< ItemType >::remove (const ItemType & *data*)

Binary tree remove function. Attempts to remove specified value from the binary tree using removeValue helper function.

Precondition

None.

Postcondition

If the value was found in the tree, it was removed and nodes were shifted if necessary to maintain integrity of the data.

Parameters

data	The value to be removed from the tree.

Returns

bool True if a value was removed, false if not.

4.2.3.24 template<class ItemType > **BinaryNode**< ItemType > ∗ **BinaryNodeTree**< ItemType >::removeValue (**BinaryNode**< ItemType > ∗ *subTreePtr,* const ItemType *target,* bool & *success*) `[protected]`

Binary tree removeValue function. Called by public remove function. Attempts to remove target value from the binary tree; will update success variable to true if operation is able to remove. Uses recursion to search the tree for the target value.

Precondition

None.

Postcondition

Value removed from the tree if found and values shifted as necessary; success variable updated to state whether the removal was successful.

Parameters

subTreePtr	The subtree to search for the value that should be removed.
target	The target value that is to be removed from the tree.
success	Bool denoting whether the value was found or not; updated to true if a removal is carried out.

Returns

BinaryNode<ItemType>∗ A pointer to the tree once the operation has been carried out.

4.2.3.25 template<class ItemType > void **BinaryNodeTree**< ItemType >::setRootData (const ItemType & *newData*)

Binary tree setRootData function. Sets the item in the root node of the binary tree to the user specified one. For an empty tree, creates the root node and stores the value in it.

Precondition

None.

Postcondition

Root node contains specified value.

Returns

None.

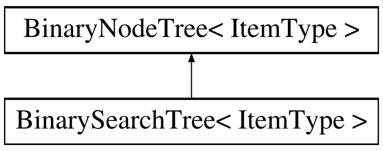
The documentation for this class was generated from the following files:

- binarySearchTree.h
- binarySearchTree.cpp

4.3 BinarySearchTree< ItemType > Class Template Reference

`#include <binarySearchTree.h>`

Inheritance diagram for BinarySearchTree< ItemType >:



Public Member Functions

- BinarySearchTree ()
- BinarySearchTree (const ItemType &rootItem)
- BinarySearchTree (const BinarySearchTree< ItemType > &tree)
- virtual ~BinarySearchTree ()
- bool isEmpty () const
- int getHeight () const
- int getNumberOfNodes () const
- ItemType getRootData () const
- void setRootData (const ItemType &newData)
- bool add (const ItemType &newEntry)
- bool remove (const ItemType &anEntry)
- void clear ()
- ItemType getEntry (const ItemType &anEntry) const
- bool contains (const ItemType &anEntry) const
- void preorderTraverse (void visit(ItemType &)) const
- void inorderTraverse (void visit(ItemType &)) const
- void postorderTraverse (void visit(ItemType &)) const
- BinarySearchTree< ItemType > & operator= (const BinarySearchTree< ItemType > &rightHandSide)

Protected Member Functions

- BinaryNode< ItemType > ∗ insertInorder (BinaryNode< ItemType > ∗subTreePtr, BinaryNode< ItemType > ∗newNodePtr)
- BinaryNode< ItemType > ∗ removeValue (BinaryNode< ItemType > ∗subTreePtr, const ItemType target, bool &success)
- BinaryNode< ItemType > ∗ removeNode (BinaryNode< ItemType > ∗nodePtr)
- BinaryNode< ItemType > ∗ removeLeftmostNode (BinaryNode< ItemType > ∗nodePtr, ItemType &inorder-Successor)
- BinaryNode< ItemType > ∗ findNode (BinaryNode< ItemType > ∗treePtr, const ItemType &target) const

4.3.1 Detailed Description

template<class ItemType>class BinarySearchTree< ItemType >

Sorted binary tree class. Uses binary nodes to build the tree. Inherits from unsorted binary tree class in order to use some of that class's protected helper methods. Does not support duplicate entries; caveat emptor (!!!). Adapted from textbook code.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 template<class ItemType > BinarySearchTree< ItemType >::BinarySearchTree ( )

BST default constructor. Tree contains no nodes, so points to null.

Precondition

None.

Postcondition

Object created with root pointing to null.

Returns

None.

4.3.2.2 template<class ItemType > BinarySearchTree< ItemType >::BinarySearchTree (const ItemType & *rootItem*)

BST parameterized constructor. Creates root node with specified data.

Precondition

None.

Postcondition

Object created with a root node containing rootItem.

Parameters

rootItem	The data the root node is to contain.

Returns

None.

4.3.2.3 template<class ItemType > BinarySearchTree< ItemType >::BinarySearchTree (const BinarySearchTree< ItemType > & *tree*)

BST copy constructor. Creates a new tree which is a copy of the tree provided.

Precondition

None.

Postcondition

BST created which is a copy of tree provided.

Parameters

tree	The tree that this object is to copy.

Returns

None.

4.3.2.4 template<class ItemType > BinarySearchTree< ItemType >::~BinarySearchTree ( ) [virtual]

BST destructor. Deallocates any dynamic memory for nodes in this tree using destroyTree function.

Precondition

None.

Postcondition

All dynamic memory for nodes in this tree deallocated.

Returns

None.

4.3.3 Member Function Documentation

4.3.3.1 template<class ItemType > bool BinarySearchTree< ItemType >::add (const ItemType & *newEntry*)

BST add function. Adds specified value to the BST as a new node in a location that will maintain the sorted property of the BST. Uses insertInorder helper function.

Precondition

None.

Postcondition

Value added to the tree; sorted property maintained.

Returns

bool Returns true, signifying successful add.

4.3.3.2 template<class ItemType > void BinarySearchTree< ItemType >::clear ( )

BST clear function. Deallocates all nodes in the tree, discarding all data, and restores tree to an empty state. Uses destroyTree helper function.

Precondition

None.

Postcondition

All nodes deallocated and all data lost; tree is returned to an empty state.

Returns

None.

4.3.3.3 template<class ItemType > bool **BinarySearchTree**< ItemType >::contains (const ItemType & *anEntry*) const

BST contains function. Searches BST for target value. Returns true if value is found, false if not. Uses findNode helper function. Search should have O(logn) binary search property.

Precondition

None.

Postcondition

Tree contents unchanged; bool signifying whether the value was found returned.

Parameters

anEntry	The value to be found in the binary tree.

Returns

bool True if the value is in the tree, false if not.

4.3.3.4 template<class ItemType > **BinaryNode**< ItemType > ∗ **BinarySearchTree**< ItemType >::findNode (**BinaryNode**< ItemType > ∗ *treePtr,* const ItemType & *target*) const [protected]

BST findNode function. Called by public getEntry and contains functions. Searches the tree for a node containing target value. Uses recursion to search the entire tree. Performs binary search so should have O(logn) property.

Precondition

None.

Postcondition

Tree contents unchanged; node containing value returned if found, nullpointer returned if value not found.

Parameters

treePtr	the tree to be searched.
target	The target value that is to be found.

Returns

BinaryNode<ItemType>∗ A pointer to the node containing the value if found, nullpointer if not found.

4.3.3.5 template<class ItemType > ItemType **BinarySearchTree**< ItemType >::getEntry (const ItemType & *anEntry*) const

BST getEntry function. Searches binary tree for target value. Returns the value if found, otherwise prints an error message to console. Uses findNode helper function.

Precondition

None.

Postcondition

Tree contents unchanged; value returned if found, otherwise error written to console.

Parameters

anEntry	The value to be found in the binary tree.

Returns

ItemType The value found in the binary tree, if the search was successful.

4.3.3.6 template<class ItemType > int BinarySearchTree< ItemType >::getHeight ( ) const

BST getHeight function. Uses getHeightHelper function to determine the height of the tree.

Precondition

None.

Postcondition

Tree contents unchanged; height returned.

Returns

int The height of the binary tree (0 if empty).

4.3.3.7 template<class ItemType > int BinarySearchTree< ItemType >::getNumberOfNodes ( ) const

BST getNumberOfNodes function. Uses getNumberOfNodesHelper function to determine the number of nodes in the tree.

Precondition

None.

Postcondition

Tree contents unchanged; number of nodes returned.

Returns

int The number of nodes in the binary tree (0 if empty).

4.3.3.8 template<class ItemType > ItemType BinarySearchTree< ItemType >::getRootData ( ) const

BST getRootData function. Binary tree must contain some data. Returns item stored in the root node of the tree. Caveat emptor using this function on an empty BST (!!!).

Precondition

The binary tree contains at least one value.

Postcondition

Tree contents unchanged; value stored in root node returned.

Returns

ItemType The item stored in the root node.

4.3.3.9 template<class ItemType > void BinarySearchTree< ItemType >::inorderTraverse (void *visitItemType &*) const

BST inorder traversal function. Visits left, then root, then right, performing an action specified in the user provided visit function. Uses preorder helper function.

Precondition

None.

Postcondition

Tree contents unchanged; visit performed for each value in the tree in specified order.

Parameters

visit	The user provided visit function, which will perform some action at each node.

Returns

None.

4.3.3.10 template<class ItemType > BinaryNode< ItemType > ∗ BinarySearchTree< ItemType >::insertInorder (BinaryNode< ItemType > ∗ *subTreePtr,* BinaryNode< ItemType > ∗ *newNodePtr*) [protected]

BST insertInorder function. Called by public add function. Adds a value to the BST in a way that maintains its sorted property. Uses recursion to find the appropriate location in the tree.

Precondition

None.

Postcondition

Value added to tree while maintaining the tree's sortedness.

Parameters

subTreePtr	The tree to which the value should be added.
newNodePtr	The node to be added to the tree.

Returns

BinaryNode<ItemType>∗ A pointer to the tree once the new value has been added.

4.3.3.11 template<class ItemType > bool BinarySearchTree< ItemType >::isEmpty ( ) const

BST isEmpty function. Checks to see if there are any nodes in the binary tree.

Precondition

None.

Postcondition

Tree contents unchanged; bool returned.

Returns

bool True if the tree is empty, false if it is not empty.

4.3.3.12 template< class ItemType > **BinarySearchTree**< ItemType > & **BinarySearchTree**< ItemType >::operator= (const **BinarySearchTree**< ItemType > & *rightHandSide*)

BST overloaded = operator. Sets left hand BST to be a copy of right hand BST using copyTree helper function.

Precondition

None.

Postcondition

Left hand BST is now a copy of right hand BST. Any nodes contained in the left hand BST before the copy deallocated and data lost.

Parameters

rightHandSide	The BST that is to be copied from.

Returns

BinarySearchTree<ItemType> The tree that now contains the copy of the right hand side tree.

4.3.3.13 template< class ItemType > void **BinarySearchTree**< ItemType >::postorderTraverse (void *visitItemType &*) const

BST postorder traversal function. Visits left, then right, then root, performing an action specified in the user provided visit function. Uses postorder helper function.

Precondition

None.

Postcondition

Tree contents unchanged; visit performed for each value in the tree in specified order.

Parameters

visit	The user provided visit function, which will perform some action at each node.

Returns

None.

4.3.3.14 template< class ItemType > void **BinarySearchTree**< ItemType >::preorderTraverse (void *visitItemType &*) const

BST preorder traversal function. Visits root, then left, then right, performing an action specified in the user provided visit function. Uses preorder helper function.

Precondition

None.

Postcondition

Tree contents unchanged; visit performed for each value in the tree in specified order.

Parameters

visit	The user provided visit function, which will perform some action at each node.

Returns

None.

4.3.3.15 template<class ItemType > bool BinarySearchTree< ItemType >::remove (const ItemType & *anEntry*)

BST remove function. Attempts to remove specified value from the binary tree using removeValue helper function while maintaining the sorted property of the BST.

Precondition

None.

Postcondition

If the value was found in the tree, it was removed and nodes were shifted if necessary to maintain integrity of the data and sortedness.

Parameters

anEntry	The value to be removed from the tree.

Returns

bool True if a value was removed, false if not.

4.3.3.16 template<class ItemType > BinaryNode< ItemType > ∗ BinarySearchTree< ItemType >::removeLeftmostNode (BinaryNode< ItemType > ∗ *nodePtr,* ItemType & *inorderSuccessor*) `[protected]`

BST removeLeftmostNode helper function. Deletes the leftmost node of the left child of target tree and sets its value to the inorderSuccessor variable. Called by removeNode function.

Precondition

Called by removeNode function in appropriate circumstance to maintain sortedness of the BST.

Postcondition

LeftmostNode deleted from the BST, inorderSuccessor value set to its data.

Parameters

nodePtr	The pointer from which the leftmost node should be deleted.
inorder-Successor	Will store the value of the leftmost node after it is deleted for use in further operations.

Returns

BinaryNode<ItemType>∗ A pointer to the tree after the operation has been conducted.

4.3.3.17 template< class ItemType > **BinaryNode**< ItemType > ∗ **BinarySearchTree**< ItemType >::removeNode (**BinaryNode**< ItemType > ∗ *nodePtr*) `[protected]`

BST removeNode function. Shifts nodes once a value to be removed has been found to maintain the integrity of the data in the tree and the tree's sortedness.

Precondition

Called by removeValue function for target node that is to be removed.

Postcondition

After all recursion, the tree will be 'fixed' to account for the removed node and sortedness maintained.

Parameters

nodePtr	Pointer to the node that is to be removed.

Returns

BinaryNode<ItemType>∗ A pointer to the tree once the operation has been carried out.

4.3.3.18 template< class ItemType > **BinaryNode**< ItemType > ∗ **BinarySearchTree**< ItemType >::removeValue (**BinaryNode**< ItemType > ∗ *subTreePtr,* const ItemType *target,* bool & *success*) `[protected]`

BST removeValue function. Called by public remove function. Attempts to remove target value from the BST while retaining sorted property and data integrity; will update success variable to true if operation finds value to remove. Uses recursion to search the tree for the target value.

Precondition

None.

Postcondition

Value removed from the tree if found and values shifted as necessary; success variable updated to state whether the removal was successful.

Parameters

subTreePtr	The subtree to search for the value that should be removed.
target	The target value that is to be removed from the tree.
success	Bool denoting whether the value was found or not; updated to true if a removal is carried out.

Returns

BinaryNode<ItemType>∗ A pointer to the tree once the operation has been carried out.

4.3.3.19 template< class ItemType > void **BinarySearchTree**< ItemType >::setRootData (const ItemType & *newData*)

BST setRootData function. If the BST is empty, creates a new node with root value. Otherwise does nothing, for fear of destroying the sorted property of the BST. Use the add function to insert the value into the tree inorder.

Precondition

None.

Postcondition

Root node contains specified value.

Returns

None.

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

- binarySearchTree.h
- binarySearchTree.cpp

Chapter 5

File Documentation

5.1 binarySearchTree.cpp File Reference

```
#include "binarySearchTree.h"
#include <iostream>
```

5.1.1 Detailed Description

CS 302 Project 6 - binary tree class implementations

Author

Patrick Austin

Date

4/13/2015

5.2 binarySearchTree.h File Reference

Classes

- class BinaryNode< ItemType >
- class BinaryNodeTree< ItemType >
- class BinarySearchTree< ItemType >

5.2.1 Detailed Description

CS 302 Project 6 - binary tree class headers

Author

Patrick Austin

Date

4/13/2015

5.3 main.cpp File Reference

```
#include "binarySearchTree.h"
#include <iostream>
#include <cstdlib>
#include <ctime>
```

Functions

- template<class ItemType >
  void my_visit (ItemType &val)
- int main ()

5.3.1 Detailed Description

CS 302 Project 6 - binary search tree test program

Author

Patrick Austin

Date

4/13/2015

5.3.2 Function Documentation

5.3.2.1 int main ( )

As per prompt, this program will...

Randomly generate 100 unique values in the range of [1-200] and insert them into a binary search tree (BST1). Print height and inorder output of the BST1 tree. Randomly generate 10 unique values in the range of [1-200] where there is an overlap with the previous values and insert them into another binary search tree (BST2). Print preorder, inorder, and postorder output of the BST2 tree. Find and remove any values of BST2 from BST1. Print height, number of nodes, and inorder output of the modified BST1 tree. Clear the binary search trees. Print whether trees are empty before and after clear operation.

5.3.2.2 template<class ItemType > void my_visit (ItemType & *val*)

Visit function for use with inorder/preorder/postorder traversals of the BSTs.

Precondition

None.

Postcondition

Visited value will be printed to console.

Parameters

val	The value of the visited item in the BST.

Returns

None.

Index