PA07 - Red Black Trees

Generated by Doxygen 1.8.11

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 RedBlackNode< itemType > Class Template Reference

Public Member Functions

- RedBlackNode ()
  The default constructor of a RedBlackNode object.
- RedBlackNode (itemType newValue, bool setIsRoot, char color)
  The parameterized constructor of a RedBlackNode object.
- ~RedBlackNode ()
  The destructor of a RedBlackNode object.
- bool IsLeftClear ()
  Checks if this node has a left child.
- bool IsRightClear ()
  Checks if this node has a right child.
- bool IsRootNode ()
  Checks if this node is the root.
- bool HasChildren ()
  Checks if this node has any children.
- void SetLeftChild (RedBlackNode *newChild)
  Sets this node's left child to the sent node.
- void SetRightChild (RedBlackNode *newChild)
  Sets this node's right child to the sent node.
- void SetLeftColor (char color)
  Sets this node's left child color to the sent color.
- void SetRightColor (char color)
  Sets this node's right child color to the sent color.
- void SetValue (itemType newValue)
  Sets this node's stored value to the sent value.
- RedBlackNode * GetLeftChild ()
  Gets the address of the left child.
- RedBlackNode * GetRightChild ()
  Gets the address of the right child.
- char GetLeftColor ()
  Gets the color of the left child.
- char GetRightColor ()
  Gets the color of the right child.
- itemType GetValue ()
  Gets this node's value.

Private Attributes

- RedBlackNode * **leftChild**
- RedBlackNode * **rightChild**
- char **leftColor**
- char **rightColor**
- itemType **value**
- bool **isRoot**

3.1.1 Constructor & Destructor Documentation

3.1.1.1 template<class itemType > RedBlackNode< itemType >::RedBlackNode ( )

The default constructor of a RedBlackNode object.

This constructor initializes values of a RedBlackNode object to default values

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

None.

Note

None.

3.1.1.2 template<class itemType > RedBlackNode< itemType >::RedBlackNode (itemType *newValue,* bool *setIsRoot,* char *color*)

The parameterized constructor of a RedBlackNode object.

This constructor initializes values of a RedBlackNode object to sent values

Algorithm None.

Parameters

in	*newValue*	The value to store in the node
in	*setIsRoot*	Tells the node if it is the root node or not
out	*None.*	

Returns

None.

Note

None.

3.1.1.3 template<class itemType > RedBlackNode< itemType >::~RedBlackNode ( )

The destructor of a RedBlackNode object.

Removes a RedBlackNode from memory.

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

None.

Note

None.

3.1.2 Member Function Documentation

3.1.2.1 template<class itemType > RedBlackNode< itemType > * RedBlackNode< itemType >::GetLeftChild ( )

Gets the address of the left child.

Returns the pointer stored in leftChild

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

Returns a pointer to the left child node.

Note

None.

3.1.2.2 template<class itemType > char RedBlackNode< itemType >::GetLeftColor ( )

Gets the color of the left child.

Returns the color of leftChild

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

Returns the color of the child as a char

Note

None.

3.1.2.3 template<class itemType > RedBlackNode< itemType > * RedBlackNode< itemType >::GetRightChild ( )

Gets the address of the right child.

Returns the pointer stored in rightChild

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

Returns a pointer to the right child node.

Note

None.

3.1.2.4 template<class itemType > char RedBlackNode< itemType >::GetRightColor ( )

Gets the color of the right child.

Returns the color of rightChild

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

Returns the color of the child as a char

Note

None.

3.1.2.5 template<class itemType > itemType RedBlackNode< itemType >::GetValue ( )

Gets this node's value.

Gets the value stored in this node.

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

Returns the value stored in this node.

Note

None.

3.1.2.6 template<class itemType > bool RedBlackNode< itemType >::HasChildren ( )

Checks if this node has any children.

Checks if this node has a left or right child by checking its pointer value.

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

None.

Note

None.

3.1.2.7 template<class itemType > bool RedBlackNode< itemType >::IsLeftClear ( )

Checks if this node has a left child.

Checks if this node has a left child by checking its pointer value.

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

None.

Note

None.

3.1.2.8 template<class itemType > bool RedBlackNode< itemType >::IsRightClear ( )

Checks if this node has a right child.

Checks if this node has a right child by checking its pointer value.

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

None.

Note

None.

3.1.2.9 template<class itemType > bool RedBlackNode< itemType >::IsRootNode ( )

Checks if this node is the root.

Checks if this node is the root node by checking the private boolean isRoot.

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

None.

Note

None.

3.1.2.10 template<class itemType > void RedBlackNode< itemType >::SetLeftChild (RedBlackNode< itemType > * *newChild*)

Sets this node's left child to the sent node.

Sets the leftChild pointer to the sent RedBlackNode pointer.

Algorithm None.

Parameters

in	*newChild*	The RedBlackNode to assign as the left child.
out	*None.*	

Returns

None.

Note

None.

3.1.2.11 template<class itemType > void RedBlackNode< itemType >::SetLeftColor (char *color*)

Sets this node's left child color to the sent color.

Sets the leftChild color to the sent color

Algorithm None.

Parameters

in	*color*	The color to set the child to
out	*None.*	

Returns

None.

Note

None.

3.1.2.12 template<class itemType > void RedBlackNode< itemType >::SetRightChild (RedBlackNode< itemType > * *newChild*)

Sets this node's right child to the sent node.

Sets the rightChild pointer to the sent RedBlackNode pointer.

Algorithm None.

Parameters

in	*newChild*	The RedBlackNode to assign as the right child.
out	*None.*	

Returns

None.

Note

None.

3.1.2.13 template<class itemType > void RedBlackNode< itemType >::SetRightColor (char *color*)

Sets this node's right child color to the sent color.

Sets the rightChild color to the sent color

Algorithm None.

Parameters

in	*color*	The color to set the child to
out	*None.*	

Returns

None.

Note

None.

3.1.2.14 template<class itemType > void RedBlackNode< itemType >::SetValue (itemType *newValue*)

Sets this node's stored value to the sent value.

Sets the node's value to the value sent as an argument.

Algorithm None.

Parameters

in	*newValue*	The new value to store in the node.
out	*None.*	

Returns

None.

Note

None.

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

- PA07/RedBlackNode.cpp

3.2 RedBlackTree< itemType > Class Template Reference

Public Member Functions

- RedBlackTree ()
  The default constructor of a RedBlackTree object.
- ~RedBlackTree ()
  The destructor of a RedBlackTree object.
- bool IsEmpty ()
  Checks if the tree is empty.
- bool Add (itemType entry)
  Adds a value to the tree.
- bool Remove (itemType target)
  Removes the target value from the tree.
- int GetHeight ()
  Gets the height of the tree.
- int GetNodeCount ()
  Gets the node count of the tree.
- void DoTraversal (int type)
  Performs a specific traversal of the tree.
- void Clear ()
  Empties the tree.
- void Print ()
  A debug function to print the tree.

Private Member Functions

- void **RotateLeft** (RedBlackNode< itemType > *subtreePtr)
- void **RotateRight** (RedBlackNode< itemType > *subtreePtr)
- void InsertFix (RedBlackNode< itemType > *subtreePtr, RedBlackNode< itemType > *targetNode)
  A function which corrects the tree.
- void RotateLeft (RedBlackNode< itemType > *subtreePtr, RedBlackNode< itemType > *targetNode)
  Does a left rotation of a selection of nodes.
- void RotateRight (RedBlackNode< itemType > *subtreePtr, RedBlackNode< itemType > *targetNode)
  Does a right rotation of a selection of nodes.
- void Insert (RedBlackNode< itemType > *subtreePtr, itemType value)
  Inserts the value into the tree.
- RedBlackNode< itemType > * GetNodeParent (RedBlackNode< itemType > *targetNode)

Gets the parent of the sent node.

- char GetNodeColor (RedBlackNode< itemType > *targetNode)

  Gets the color of the sent node.
- int InorderTraverse (RedBlackNode< itemType > *subtreePtr)

  Does an inorder traversal of the tree.
- int CountChildren (RedBlackNode< itemType > *subtreePtr)

  Counts the nodes in the tree.
- int CountHeight (RedBlackNode< itemType > *subtreePtr)

  Finds the longest branch of the tree.
- void DebugPrint (RedBlackNode< itemType > *subtreePtr)

  A debug function to print the tree.

Private Attributes

- RedBlackNode< itemType > * **rootPtr**
- int **nodeCount**

3.2.1 Constructor & Destructor Documentation

3.2.1.1 template<class itemType > RedBlackTree< itemType >::RedBlackTree ( )

The default constructor of a RedBlackTree object.

This constructor initializes values of a RedBlackTree object to default values

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

None.

Note

None.

3.2.1.2 template<class itemType > RedBlackTree< itemType >::~RedBlackTree ( )

The destructor of a RedBlackTree object.

This removes the RedBlackTree from memory

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

None.

Note

None.

3.2.2 Member Function Documentation

3.2.2.1 template<class itemType > bool RedBlackTree< itemType >::Add (itemType *entry*)

Adds a value to the tree.

Adds the sent value to the tree

Algorithm Recursively finds the proper position for the new node

Parameters

in	*entry*	The value to store in the tree
out	*None.*	

Returns

Returns a bool signifying if the node could be added, always true

Note

None.

3.2.2.2 template<class itemType > void RedBlackTree< itemType >::Clear ( )

Empties the tree.

Clears the tree by deleting the root and then setting it to null.

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

None.

Note

None.

3.2.2.3 template<class itemType > int RedBlackTree< itemType >::CountChildren (RedBlackNode< itemType > * *subtreePtr*) [private]

Counts the nodes in the tree.

Runs through the tree and counts each node.

Algorithm none.

Parameters

in	*subtreePtr*	The pointer to the tree to traverse
out	*None.*	

Returns

Returns the count of the nodes.

Note

None.

3.2.2.4 template<class itemType > int RedBlackTree< itemType >::CountHeight (RedBlackNode< itemType > * *subtreePtr*) [private]

Finds the longest branch of the tree.

Runs through the tree and returns the height of the longest branch.

Algorithm none.

Parameters

in	*subtreePtr*	The pointer to the tree to traverse
out	*None.*	

Returns

Returns the height of the tree.

Note

None.

3.2.2.5 template<class itemType > void RedBlackTree< itemType >::DebugPrint (RedBlackNode< itemType > * *subtreePtr*) [private]

A debug function to print the tree.

Prints the tree, used for debugging.

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

None.

Note

None.

3.2.2.6 template<class itemType > void RedBlackTree< itemType >::DoTraversal (int *type*)

Performs a specific traversal of the tree.

Expects a value 0-2 which picks the type of traversal to do.

Algorithm None.

Parameters

in	*type*	An int which is the type of traversal to do. 0 = Pre, 1 = In, 2 = Post
out	*None.*	

Returns

None.

Note

None.

3.2.2.7 template<class itemType > int RedBlackTree< itemType >::GetHeight ( )

Gets the height of the tree.

Gets the height of the tree by counting the longest branch.

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

Returns an int which is the height of the tree.

Note

None.

3.2.2.8 template<class itemType > char RedBlackTree< itemType >::GetNodeColor (RedBlackNode< itemType > * *targetNode*) [private]

Gets the color of the sent node.

Gets the color of the sent node by checking the parent

Algorithm None.

Parameters

in	*targetNode*	Pointer to the node to get the color of
out	*None.*	

Returns

None.

Note

Color is stored in the parent, an individual node does not know its own color

3.2.2.9 template<class itemType > int RedBlackTree< itemType >::GetNodeCount ( )

Gets the node count of the tree.

Gets the node count of the tree by counting each node.

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

Returns an int which is the node count of the tree.

Note

None.

3.2.2.10 template<class itemType > RedBlackNode< itemType > * RedBlackTree< itemType >::GetNodeParent (RedBlackNode< itemType > * *targetNode*) [private]

Gets the parent of the sent node.

Traverses the tree to find the parent of the sent node

Algorithm None.

Parameters

in	*targetNode*	Pointer to the node to find the parent of
out	*None.*	

Returns

Returns a pointer to the parent node

Note

None.

3.2.2.11 template<class itemType > int RedBlackTree< itemType >::InorderTraverse (RedBlackNode< itemType > * *subtreePtr*) [private]

Does an inorder traversal of the tree.

Traverses the tree by printing the left, current root, and then the right child.

Algorithm None.

Parameters

in	*subtreePtr*	Pointer to the current subtree.
out	*None.*	

Returns

None.

Note

None.

3.2.2.12 template<class itemType > void RedBlackTree< itemType >::Insert (RedBlackNode< itemType > * *subtreePtr,* itemType *value*) [private]

Inserts the value into the tree.

Creates a new node with the sent value and adds it to the tree, calls InsertFix after

Algorithm None.

Parameters

in	*subtreePtr*	Pointer to the tree
in	*value*	The value to be added to the tree
out	*None.*	

Returns

None.

Note

None.

3.2.2.13 template<class itemType > void RedBlackTree< itemType >::InsertFix (RedBlackNode< itemType > * *subtreePtr,* RedBlackNode< itemType > * *targetNode*) [private]

A function which corrects the tree.

Called after an insertion is made, this function automatically corrects the tree

Algorithm None.

Parameters

in	*subtreePtr*	Pointer to the tree
in	*targetNode*	Pointer to the new node to start corrections at
out	*None.*	

Returns

None.

Note

None.

3.2.2.14 template<class itemType > bool RedBlackTree< itemType >::IsEmpty ( )

Checks if the tree is empty.

Checks if the tree is empty by checking the root pointer

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

Returns a bool signifying whether or not the tree is empty

Note

None.

3.2.2.15 template<class itemType > void RedBlackTree< itemType >::Print ( )

A debug function to print the tree.

Prints the tree, used for debugging.

Algorithm None.

Parameters

in	*None.*	
out	*None.*	

Returns

None.

Note

None.

3.2.2.16 template<class itemType > bool RedBlackTree< itemType >::Remove (itemType *target*)

Removes the target value from the tree.

Seaches for the target value and removes it if it is found

Algorithm Recursively finds the value and then removes it

Parameters

in	*target*	The value to remove from the tree
out	*None.*	

Returns

Returns a bool signifying if the node could be removed, false if the value doesn't exist

Note

None.

3.2.2.17 template<class itemType > void RedBlackTree< itemType >::RotateLeft (RedBlackNode< itemType > * *subtreePtr,* RedBlackNode< itemType > * *targetNode*) [private]

Does a left rotation of a selection of nodes.

Called by insertFix(), this function performs a left rotation of some nodes

Algorithm None.

Parameters

in	*subtreePtr*	Pointer to the tree
in	*targetNode*	Pointer to the node to do the rotation from
out	*None.*	

Returns

None.

Note

None.

3.2.2.18 template<class itemType > void RedBlackTree< itemType >::RotateRight (RedBlackNode< itemType > * *subtreePtr,* RedBlackNode< itemType > * *targetNode*) [private]

Does a right rotation of a selection of nodes.

Called by insertFix(), this function performs a right rotation of some nodes

Algorithm None.

Parameters

in	*subtreePtr*	Pointer to the tree
in	*targetNode*	Pointer to the node to do the rotation from
out	*None.*	

Returns

None.

Note

None.

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

- PA07/RedBlackTree.cpp

4 File Documentation

4.1 PA07/PA07.cpp File Reference

This is the main file to run the tree.

```
#include <iostream>
#include "RedBlackTree.cpp"
#include <cstdlib>
#include <time.h>
```

Functions

- void GenerateUniqueValues (int *destination, int amount)
  Fills an array with unique values.
- int **main** ()

4.1.1 Detailed Description

This is the main file to run the tree.

Author

Bryce Monaco

This file runs the tree and performs the required operations.

Version

1.0

Note

GenerateUniqueValues() is adapted from PA06

4.1.2 Function Documentation

4.1.2.1 void GenerateUniqueValues (int * *destination,* int *amount*)

Fills an array with unique values.

Fills an array with unique random values.

Algorithm Checks if the value has already been generated and then stores it or regenerates it.

Parameters

in	*destination*	The integer array to store the data in
in	*amount*	The number of values to generate, must be the size of the array
out	*None.*	

Returns

None.

Note

Adapted from code written for PA06 in PA06.cpp

4.2 PA07/RedBlackNode.cpp File Reference

This is the header and implmentation of RedBlackNode.

```
#include <iostream>
```

Classes

- class RedBlackNode< itemType >

4.2.1 Detailed Description

This is the header and implmentation of RedBlackNode.

Author

Bryce Monaco

This file contains the header and implementation of RedBlackNode

Version

1.0

Note

Header and implementation are in one file to fix some templating issues. This code is mostly copied from PA06's LeafNode.cpp and adapted for RB Trees.

4.3 PA07/RedBlackTree.cpp File Reference

This is the header and implmentation of RedBlackTree.

```
#include <iostream>
#include "RedBlackNode.cpp"
```

Classes

- class RedBlackTree< itemType >

4.3.1 Detailed Description

This is the header and implmentation of RedBlackTree.

Author

Bryce Monaco

This file contains the header and implementation of RedBlackTree

Version

1.0

Note

Header and implementation are in one file to fix some templating issues.

Index