PA06 - Binary Seach Trees

Generated by Doxygen 1.8.11

Contents

1	Clas	s Index		1
	1.1	Class L	.ist	1
2	File	Index		2
	2.1	File Lis	t	2
3	Clas	s Docur	mentation	2
	3.1	Binary	SearchTree< itemType > Class Template Reference	2
		3.1.1	Constructor & Destructor Documentation	3
		3.1.2	Member Function Documentation	4
	3.2	LeafNo	de< itemType > Class Template Reference	13
		3.2.1	Constructor & Destructor Documentation	14
		3.2.2	Member Function Documentation	16
4	File	Docume	entation	21
	4.1	PA06/E	BinarySearchTree.cpp File Reference	21
		4.1.1	Detailed Description	22
	4.2	PA06/L	eafNode.cpp File Reference	22
		4.2.1	Detailed Description	22
	4.3	PA06/F	PA06.cpp File Reference	23
		4.3.1	Detailed Description	23
		4.3.2	Function Documentation	23
Inc	dex			27
1	Cla	ass Ind	ex	
1.1	ı CI	ass List		
1.1	ı Cl	ass List		
He	re are	e the clas	sses, structs, unions and interfaces with brief descriptions:	
	Bina	ırySearc	hTree < itemType >	2
	LeafNode < itemType >			

2	F	ile	Inc	lex
---	---	-----	-----	-----

2.1 File List

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

PA06/BinarySearchTree.cpp	
This is the header and implmentation of BinarySearchTree	21
PA06/LeafNode.cpp	
This is the header and implmentation of LeafNode	22
PA06/PA06.cpp	
This is the main file to run the trees	23

3 Class Documentation

3.1 BinarySearchTree < itemType > Class Template Reference

Public Member Functions

• BinarySearchTree ()

The default constructor of a BinarySearchTree object.

∼BinarySearchTree ()

The destructor of a BinarySearchTree 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

LeafNode< itemType > * PlaceNode (LeafNode< itemType > *subtreePtr, LeafNode< itemType > *new← Node)

Places a new node at the first available and proper position.

LeafNode< itemType > * RemoveValue (LeafNode< itemType > *subtreePtr, itemType target, bool &is←
Successful)

removes the node with the value

LeafNode< itemType > * RemoveNode (LeafNode< itemType > *nodePtr)

removes the sent node

LeafNode< itemType > * RemoveLeftmostNode (LeafNode< itemType > *nodePtr, itemType &successor ← Value)

removes the leftmost node in a tree

void PreorderTraverse (LeafNode < itemType > *subtreePtr)

Does a preorder traversal of the tree.

void InorderTraverse (LeafNode< itemType > *subtreePtr)

Does an inorder traversal of the tree.

void PostorderTraverse (LeafNode < itemType > *subtreePtr)

Does a postorder traversal of the tree.

int CountChildren (LeafNode < itemType > *subtreePtr)

Counts the nodes in the tree.

int CountHeight (LeafNode < itemType > *subtreePtr)

Finds the longest branch of the tree.

void DebugPrint (LeafNode< itemType > *subtreePtr)

A debug function to print the tree.

Private Attributes

- LeafNode < itemType > * rootPtr
- int nodeCount
- 3.1.1 Constructor & Destructor Documentation
- ${\tt 3.1.1.1} \quad template < {\tt class\ itemType} > {\tt BinarySearchTree} < {\tt itemType} > {\tt ::BinarySearchTree} (\quad)$

The default constructor of a BinarySearchTree object.

This constructor initializes values of a BinarySearchTree object to default values

Algorithm None.

Parameters

in	None.	
out	None.	

Returns
None.
Note
None.
3.1.1.2 template < class itemType > BinarySearchTree < itemType >::~BinarySearchTree ()
The destructor of a BinarySearchTree object.
This removes the BinarySearchTree from memory
Algorithm None.
Parameters in None. out None.
Returns None.
Note
None.
3.1.2 Member Function Documentation
${\it 3.1.2.1 template}{<} {\it class itemType} > {\it bool BinarySearchTree}{<} {\it itemType} > {\it ::Add (itemType entry)}$
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.	

	- 4	L		
к	ρı	ш	r	n۹

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

Note

None.

3.1.2.2 template < class itemType > void BinarySearchTree < 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.

3.1.2.3 template < class itemType > int BinarySearchTree < itemType > ::CountChildren (LeafNode < 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.

3.1.2.4 template < class itemType > int BinarySearchTree < itemType > ::CountHeight (LeafNode < 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.

3.1.2.5 template < class itemType > void BinarySearchTree < itemType > ::DebugPrint (LeafNode < 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.
$\textbf{3.1.2.6} template < class \ itemType > void \ BinarySearchTree < itemType > :: DoTraversal \ (\ int \ \mathit{type} \)$
Performs a specific traversal of the tree.
Expects a value 0-2 which picks the type of traversal to do.
Algorithm None.
Paramatana
Parameters in type An int which is the type of traversal to do. 0 = Pre, 1 = In, 2 = Post
out None.
Returns
None.
Note
Note None.
None.
3.1.2.7 template < class itemType > int BinarySearchTree < 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.

8	CONTENTS
Note	
None.	
${\it 3.1.2.8 template}{<} {\it class itemType} > {\it int BinarySearchTree}{<} {\it itemType} > {\it ::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.	
3.1.2.9 template <class itemtype=""> void BinarySearchTree< itemType >::InorderTraverse(LosubtreePtr) [private]</class>	eafNode< itemType > *
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 <i>None.</i>	

in	subtreePtr	Pointer to the current subtree.
out	None.	

D	A+1	LEG	

None.

Note

None.

3.1.2.10 template < class itemType > bool BinarySearchTree < itemType >::lsEmpty ()

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.

```
3.1.2.11 template < class itemType > LeafNode < itemType > * BinarySearchTree < itemType > ::PlaceNode ( LeafNode < itemType > * subtreePtr, LeafNode < itemType > * newNode ) [private]
```

Places a new node at the first available and proper position.

Goes down the tree until it finds a position where the node can be stored.

Algorithm Recursively traverses the tree to find a valid position for the new node.

Parameters

in	subTreePtr	The pointer to the current subtree
in	newNode	The pointer to the new node to be placed
out	None.	

Returns

Returns a pointer to the newly placed node.

Note

if the entry data type does not have a definition for '>' and '<' that the function will fail

3.1.2.12	template <class< th=""><th>itemType $>$ void BinarySearchTree$<$ itemType $>$::PostorderTraverse (LeafNode$<$ itemType $>$</th></class<>	itemType $>$ void BinarySearchTree $<$ itemType $>$::PostorderTraverse (LeafNode $<$ itemType $>$
	* subtreePtr)	[private]

Does a postorder traversal of the tree.

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

Algorithm None.

Parameters

in	subtreePtr	Pointer to the current subtree.
out	None.	

Returns

None.

Note

None.

3.1.2.13 template < class itemType > void BinarySearchTree < itemType > ::PreorderTraverse (LeafNode < itemType > * subtreePtr) [private]

Does a preorder traversal of the tree.

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

Algorithm None.

Parameters

in	subtreePtr	Pointer to the current subtree.
out	None.	

	-4		
n	еп	ur	HS.

None.

Note

None.

3.1.2.14 template < class itemType > void BinarySearchTree < 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.
None.
${\it 3.1.2.15} {\it template}{<} {\it class itemType} > {\it bool BinarySearchTree}{<} {\it itemType} > {\it ::Remove (itemType target)}$
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
Note None.

3.1.2.16	template < class itemType > LeafNode < itemType > * BinarySeathers = Continuous Contin	archTree < itemType >::RemoveLeftmostNode
	LeafNode < itemType > * nodePtr, itemType & successorValue)	[private]

removes the leftmost node in a tree

Removes the leftmost node in the sent branch

Algorithm none.

Parameters

in	nodePtr	The pointer to the node to remove
in	successorValue	The value of the previous node
out	None.	

Returns

Returns a pointer to the subtree

Note

None.

```
3.1.2.17 template < class itemType > LeafNode < itemType > * BinarySearchTree < itemType >::RemoveNode ( LeafNode < itemType > * nodePtr ) [private]
```

removes the sent node

Removes the sent node and fixes the children

Algorithm none.

Parameters

in	nodePtr	The pointer to the node to remove
out	None.	

Returns

Returns a pointer to the subtree

Note

None.

removes the node with the value

Removes the node with the specific value and then fixes the leaves.

Algorithm none.

Parameters

in	subTreePtr	The pointer to the current subtree
in	target	The value to remove
in	isSuccessful	The bool to signify if the node could be found and removed
out	None.	

Returns

Returns a pointer to the subtree

Note

None.

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

• PA06/BinarySearchTree.cpp

3.2 LeafNode < itemType > Class Template Reference

Public Member Functions

• LeafNode ()

The default constructor of a LeafNode object.

LeafNode (itemType newValue, bool setIsRoot)

The parameterized constructor of a LeafNode object.

• \sim LeafNode ()

The destructor of a LeafNode 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 (LeafNode *newChild)

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

void SetRightChild (LeafNode *newChild)

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

void SetValue (itemType newValue)

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

LeafNode * GetLeftChild ()

Gets the address of the left child.

LeafNode * GetRightChild ()

Gets the address of the right child.

• itemType GetValue ()

Gets this node's value.

Private Attributes

- LeafNode * leftChild
- LeafNode * rightChild
- itemType value
- bool isRoot
- 3.2.1 Constructor & Destructor Documentation
- 3.2.1.1 template < class itemType > LeafNode < itemType >::LeafNode ()

The default constructor of a LeafNode object.

This constructor initializes values of a LeafNode object to default values

Algorithm None.

Parameters

in	None.	
out	None.	

Returns

None.

Note

None.

3.2.1.2 template < class itemType > LeafNode < itemType >::LeafNode (itemType newValue, bool setIsRoot)

The parameterized constructor of a LeafNode object.

This constructor initializes values of a LeafNode object to sent values

3.2 LeafNode< itemType > Class Template Refer	erence
---	--------

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.	

Ret	ur	ns
HICK	uı	пə

None.

Note

None.

3.2.1.3 template < class itemType > LeafNode < itemType >:: \sim LeafNode ()

The destructor of a LeafNode object.

Removes a LeafNode from memory.

Algorithm None.

Parameters

in	None.	
out	None.	

Returns

None.

Note

None.

3.2.2 Member Function Documentation

 $\textbf{3.2.2.1} \quad \textbf{template} < \textbf{class itemType} > \textbf{LeafNode} < \textbf{itemType} > * \textbf{LeafNode} < \textbf{itemType} > :: \textbf{GetLeftChild (} \quad \textbf{)}$

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.

 ${\tt 3.2.2.2} \quad {\tt template}{<} {\tt class\ itemType} > {\tt LeafNode}{<}\ {\tt itemType} > {\tt *LeafNode}{<}\ {\tt itemType} > {\tt ::GetRightChild}\ (\quad)$

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.

3.2.2.3 template < class itemType > itemType LeafNode < itemType >::GetValue ()

Gets this node's value.

Gets the value stored in this node.

Algorithm None.

Parameters

in	None.	
out	None.	

Generated by Doxygen

_			
D	Λŧ:	IPP	0

Returns the value stored in this node.

Note

None.

3.2.2.4 template < class itemType > bool LeafNode < 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.

3.2.2.5 template < class itemType > bool LeafNode < 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.
3.2.2.6 template < class itemType > bool LeafNode < 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
Note None.
${\it 3.2.2.7} {\it template}{<} {\it class itemType} > {\it bool LeafNode}{<} {\it itemType} > {\it ::} {\it lsRootNode} (\ \)$
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.

3.2.2.8 to	emplate <cla< th=""><th>${\sf ss} \; {\sf itemType} > {\sf void} \; {\sf LeafNode} < {\sf itemType} > {\it ::} {\sf SetLeftChild} \; (\; \; {\sf LeafNode} < {\sf itemType} > * \; {\it newChild}$</th><th>)</th></cla<>	${\sf ss} \; {\sf itemType} > {\sf void} \; {\sf LeafNode} < {\sf itemType} > {\it ::} {\sf SetLeftChild} \; (\; \; {\sf LeafNode} < {\sf itemType} > * \; {\it newChild} $)
Sets this	node's left	child to the sent node.	
Sets the	leftChild po	pinter to the sent LeafNode pointer.	
Algorithm N	None.		
Parameter	rs		
in	newChild	The LeafNode to assign as the left child.	
out	None.		
Returns			
No	ne.		
Note			
No	ne.		
3.2.2.9 to	emplate <cla< td=""><td>${\sf ss}$ itemType $>$ void LeafNode $<$ itemType $>$::SetRightChild (LeafNode $<$ itemType $>$ * ${\it newChild}$</td><td>ld</td></cla<>	${\sf ss}$ itemType $>$ void LeafNode $<$ itemType $>$::SetRightChild (LeafNode $<$ itemType $>$ * ${\it newChild}$	ld
Sets this	node's righ	nt child to the sent node.	
Sets the	rightChild p	pointer to the sent LeafNode pointer.	
Algorithm N	None.		
Parameter	rs		
in	newChild	The LeafNode to assign as the right child.	
out	None.		
Returns			
No	ne.		
Note			
Note			
No	ne.		

4 File Documentation 21

3.2.2.10 template < class itemType > void LeafNode < 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:

PA06/LeafNode.cpp

4 File Documentation

4.1 PA06/BinarySearchTree.cpp File Reference

This is the header and implmentation of BinarySearchTree.

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

Classes

 $\bullet \ \ {\it class BinarySearchTree}{< i temType >}$

4.1.1 Detailed Description

This is the header and implmentation of BinarySearchTree.

Author

Bryce Monaco

This file contains the header and implementation of BinarySearchTree

Version

1.0

Note

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

4.2 PA06/LeafNode.cpp File Reference

This is the header and implmentation of LeafNode.

```
#include <iostream>
```

Classes

class LeafNode < itemType >

4.2.1 Detailed Description

This is the header and implmentation of LeafNode.

Author

Bryce Monaco

This file contains the header and implementation of LeafNode

Version

1.0

Note

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

4.3 PA06/PA06.cpp File Reference

This is the main file to run the trees.

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

Functions

void GenerateUniqueValues (int *destination, int amount)

Fills an array with unique values.

• int GenerateUniqueOverlapValues (int *destination, int *mainData, int amountMain, int amountSecond, int overlaps)

Fills an array with unique values with some overlap of another array.

void DoTraversals (BinarySearchTree< int > *sentTree)

Does all three traversals with the sent tree.

• int main ()

4.3.1 Detailed Description

This is the main file to run the trees.

Author

Bryce Monaco

This file runs the trees and performs the required operations.

Version

1.0

Note

None.

4.3.2 Function Documentation

4.3.2.1 void DoTraversals (BinarySearchTree < int > * sentTree)

Does all three traversals with the sent tree.

Does all three traversals wit hthe sent tree and formats the output

Algorithm None.

Parameters

in	sentTree	The tree to do the traversals on.
out	None.	

١

None.

Note

None.

4.3.2.2 int GenerateUniqueOverlapValues (int * destination, int * mainData, int amountMain, int amountSecond, int overlaps)

Fills an array with unique values with some overlap of another array.

Fills an array with unique random values with at least a certain amount of overlaps

Algorithm Checks if there are a certain amount of overlapping values, if there are it stores, if not it regenerates

Parameters

in	destination	The integer array to store the data in
in	mainData	The integer array of the main data to check for overlaps
in	amountMain	The size of the main array
in	amountSecond	The size of the overlap array and the number of values to generate
in	overlaps	The minimum number of overlapping values to generate
out	None.	

Returns

Returns the number of overlaps generated, will be greater than or equal to the sent overlaps value

Note

None.

4.3.2.3 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

None.

Index

~BinarySearchTree	LeafNode, 17
BinarySearchTree, 4	GetValue
~LeafNode	LeafNode, 17
LeafNode, 16	
	HasChildren
Add	LeafNode, 18
BinarySearchTree, 4	
	InorderTraverse
BinarySearchTree	BinarySearchTree, 8
\sim BinarySearchTree, 4	IsEmpty
Add, 4	BinarySearchTree, 8
BinarySearchTree, 3	IsLeftClear
Clear, 5	LeafNode, 18
CountChildren, 5	IsRightClear
CountHeight, 6	LeafNode, 19
DebugPrint, 6	IsRootNode
DoTraversal, 7	LeafNode, 19
GetHeight, 7	
GetNodeCount, 8	LeafNode
InorderTraverse, 8	\sim LeafNode, 16
IsEmpty, 8	GetLeftChild, 16
PlaceNode, 9	GetRightChild, 17
PostorderTraverse, 9	GetValue, 17
PreorderTraverse, 10	HasChildren, 18
Print, 10	IsLeftClear, 18
Remove, 11	IsRightClear, 19
RemoveLeftmostNode, 11	IsRootNode, 19
RemoveNode, 12	LeafNode, 14
RemoveValue, 12	SetLeftChild, 19
BinarySearchTree< itemType >, 2	SetRightChild, 20
	SetValue, 20
Clear	LeafNode< itemType >, 13
BinarySearchTree, 5	DA00
CountChildren	PA06.cpp
BinarySearchTree, 5	DoTraversals, 23
CountHeight	GenerateUniqueOverlapValues, 24
BinarySearchTree, 6	GenerateUniqueValues, 24
Dalam Drint	PA06/BinarySearchTree.cpp, 21
DebugPrint Carach Trace C	PA06/LeafNode.cpp, 22
BinarySearchTree, 6	PA06/PA06.cpp, 23
DoTraversal	PlaceNode
BinarySearchTree, 7	BinarySearchTree, 9
DoTraversals	PostorderTraverse
PA06.cpp, 23	BinarySearchTree, 9 PreorderTraverse
GenerateUniqueOverlapValues	
PA06.cpp, 24	BinarySearchTree, 10 Print
GenerateUniqueValues	-
PA06.cpp, 24	BinarySearchTree, 10
• •	Remove
GetHeight BinarySearchTree, 7	BinarySearchTree, 11
GetLeftChild	RemoveLeftmostNode
LeafNode, 16	BinarySearchTree, 11
GetNodeCount	RemoveNode
BinarySearchTree, 8	BinarySearchTree, 12
GetRightChild	RemoveValue
Gett light Offilia	i temove value

28 INDEX

BinarySearchTree, 12

SetLeftChild

LeafNode, 19

SetRightChild

LeafNode, 20

SetValue

LeafNode, 20