

## CS 250 Project 3: Binary Search Tree

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Project 3: Binary Search Tree</b>	<b>1</b>
<b>2</b>	<b>Hierarchical Index</b>	<b>3</b>
2.1	Class Hierarchy . . . . .	3
<b>3</b>	<b>Class Index</b>	<b>5</b>
3.1	Class List . . . . .	5
<b>4</b>	<b>File Index</b>	<b>7</b>
4.1	File List . . . . .	7
<b>5</b>	<b>Class Documentation</b>	<b>9</b>
5.1	BinarySearchTree< TK, TD > Class Template Reference . . . . .	9
5.1.1	Detailed Description . . . . .	10
5.1.2	Constructor & Destructor Documentation . . . . .	11
5.1.2.1	BinarySearchTree() . . . . .	11
5.1.2.2	~BinarySearchTree() . . . . .	11
5.1.3	Member Function Documentation . . . . .	11
5.1.3.1	Contains(const TK &key) . . . . .	11
5.1.3.2	Delete(const TK &key) . . . . .	11
5.1.3.3	DeleteNode_LeftChild(Node< TK, TD > *deleteMe, Node< TK, TD > *parent, bool isLeftNode) . . . . .	12
5.1.3.4	DeleteNode_NoChildren(Node< TK, TD > *deleteMe, Node< TK, TD > *parent, bool isLeftNode) . . . . .	12
5.1.3.5	DeleteNode_RightChild(Node< TK, TD > *deleteMe, Node< TK, TD > *parent, bool isLeftNode) . . . . .	12

5.1.3.6	DeleteNode_TwoChildren(Node< TK, TD > *deleteMe, Node< TK, TD > *parent, bool isLeftNode)	13
5.1.3.7	FindNode(const TK &key)	13
5.1.3.8	FindParentOfNode(const TK &key)	14
5.1.3.9	GetCount()	14
5.1.3.10	GetData(const TK &key)	14
5.1.3.11	GetHeight()	14
5.1.3.12	GetHeight(Node< TK, TD > *ptrCurrent)	15
5.1.3.13	GetInOrder()	15
5.1.3.14	GetInOrder(Node< TK, TD > *ptrCurrent, stringstream &stream)	15
5.1.3.15	GetMax(Node< TK, TD > *ptrCurrent)	15
5.1.3.16	GetMaxKey()	16
5.1.3.17	GetMin(Node< TK, TD > *ptrCurrent)	16
5.1.3.18	GetMinKey()	16
5.1.3.19	GetPostOrder()	16
5.1.3.20	GetPostOrder(Node< TK, TD > *ptrCurrent, stringstream &stream)	16
5.1.3.21	GetPreOrder()	17
5.1.3.22	GetPreOrder(Node< TK, TD > *ptrCurrent, stringstream &stream)	17
5.1.3.23	Push(const TK &newKey, const TD &newData)	17
5.1.3.24	RecursiveContains(const TK &key, Node< TK, TD > *ptrCurrent)	18
5.1.3.25	RecursiveFindNode(const TK &key, Node< TK, TD > *ptrCurrent)	18
5.1.3.26	RecursivePush(const TK &newKey, const TD &newData, Node< TK, TD > *ptrCurrent)	18
5.1.4	Friends And Related Function Documentation	18
5.1.4.1	Tester	18
5.1.5	Member Data Documentation	18
5.1.5.1	m_nodeCount	18
5.1.5.2	m_ptrRoot	19
5.2	CarData Struct Reference	19
5.2.1	Member Data Documentation	19
5.2.1.1	make	19

5.2.1.2	model	19
5.2.1.3	price	19
5.3	CarProgram Class Reference	19
5.3.1	Member Function Documentation	20
5.3.1.1	LoadData(const string &carFile)	20
5.3.1.2	MainLoop()	20
5.3.1.3	PrintStats(const string &outFile, const string &orderFile)	20
5.3.1.4	SaveData(const string &carFile)	20
5.3.1.5	Start()	20
5.3.2	Member Data Documentation	20
5.3.2.1	m_data_linear	20
5.3.2.2	m_data_tree	20
5.4	LinkedList< TK, TD > Class Template Reference	20
5.4.1	Constructor & Destructor Documentation	21
5.4.1.1	LinkedList()	21
5.4.1.2	~LinkedList()	21
5.4.2	Member Function Documentation	21
5.4.2.1	Clear()	21
5.4.2.2	Contains(TD item)	21
5.4.2.3	Display()	21
5.4.2.4	GetFirst()	21
5.4.2.5	GetItemAtPosition(int index)	21
5.4.2.6	GetItemWithKey(const TK &key)	21
5.4.2.7	GetLast()	21
5.4.2.8	IsEmpty()	21
5.4.2.9	PopBack()	22
5.4.2.10	PopFront()	22
5.4.2.11	PushBack(const TK &newKey, const TD &newData)	22
5.4.2.12	PushFront(const TK &newKey, const TD &newData)	22
5.4.2.13	Size()	22

5.4.3	Member Data Documentation . . . . .	22
5.4.3.1	m_size . . . . .	22
5.4.3.2	ptrEnd . . . . .	22
5.4.3.3	ptrFront . . . . .	22
5.5	Node< TK, TD > Class Template Reference . . . . .	22
5.5.1	Detailed Description . . . . .	23
5.5.2	Constructor & Destructor Documentation . . . . .	23
5.5.2.1	Node() . . . . .	23
5.5.2.2	Node(TK newKey, TD newData) . . . . .	23
5.5.2.3	~Node() . . . . .	23
5.5.3	Member Function Documentation . . . . .	23
5.5.3.1	IsNotTree() . . . . .	23
5.5.4	Member Data Documentation . . . . .	23
5.5.4.1	data . . . . .	23
5.5.4.2	isTree . . . . .	23
5.5.4.3	key . . . . .	23
5.5.4.4	ptrLeft . . . . .	24
5.5.4.5	ptrRight . . . . .	24
5.6	Tester Class Reference . . . . .	24
5.6.1	Detailed Description . . . . .	24
5.6.2	Constructor & Destructor Documentation . . . . .	25
5.6.2.1	Tester() . . . . .	25
5.6.3	Member Function Documentation . . . . .	25
5.6.3.1	Test_Contains() . . . . .	25
5.6.3.2	Test_Delete() . . . . .	25
5.6.3.3	Test_FindNode() . . . . .	25
5.6.3.4	Test_FindParentOfNode() . . . . .	25
5.6.3.5	Test_GetCount() . . . . .	25
5.6.3.6	Test_GetHeight() . . . . .	25
5.6.3.7	Test_GetInOrder() . . . . .	25
5.6.3.8	Test_GetMax() . . . . .	25
5.6.3.9	Test_GetPostOrder() . . . . .	25
5.6.3.10	Test_GetPreOrder() . . . . .	25
5.6.3.11	Test_Push() . . . . .	25
5.7	Timer Class Reference . . . . .	25
5.7.1	Member Function Documentation . . . . .	26
5.7.1.1	GetElapsedMilliseconds() . . . . .	26
5.7.1.2	GetElapsedSeconds() . . . . .	26
5.7.1.3	Start() . . . . .	26
5.7.2	Member Data Documentation . . . . .	26
5.7.2.1	m_startTime . . . . .	26

<b>6 File Documentation</b>	<b>27</b>
6.1 BinarySearchTree.hpp File Reference	27
6.2 car-data.txt File Reference	27
6.3 CarData.hpp File Reference	27
6.4 CarProgram.cpp File Reference	27
6.5 CarProgram.hpp File Reference	28
6.6 data-car-makes.txt File Reference	28
6.7 LinkedList.hpp File Reference	28
6.8 main.cpp File Reference	28
6.8.1 Function Documentation	28
6.8.1.1 main()	28
6.9 main.md File Reference	28
6.10 Node.hpp File Reference	28
6.11 program_main.cpp File Reference	29
6.11.1 Function Documentation	29
6.11.1.1 main()	29
6.12 Tester.hpp File Reference	29
6.12.1 Macro Definition Documentation	29
6.12.1.1 _TESTER_HPP	29
6.13 Timer.hpp File Reference	29
<b>Index</b>	<b>31</b>





# Chapter 1

## Project 3: Binary Search Tree

### Viewing the documentation

Click on the **Classes** tab to view files, classes, and their documentation. Click on the **Binary Search Tree** link to view a list of all functions.

### About

For this project, you will read the instructions from the Doxygen file (or the comments in the code) to complete the program.

Use the **unit tests** provided to test your Binary Search Tree as you are implementing features.

### Main menu

When you first start the program, you can choose to run the tests, the program, or exit.

```
1 -----
2 | MAIN MENU |
3 -----
4
5 1. Tests
6 2. Program
7 3. Exit
8
9 >>
```

Just run the tests while you're working on completing the Binary Search Tree.

### Input/output files

When the program is running, it will ask what files to use for the input and output files. Three files are provided:

`car-data-big.txt`, `car-data-medium.txt`, and `car-data-small.txt`

You can also create an empty file, then use the program to add new items in, one at a time.

## Data

In the tree, each node will have a key and a data value. The key is the car's price (a float), and the data is the `CarData` struct with its make, model, and price.

## Suggested order...

Delete is going to be the most difficult function to implement. Other functions are recursive. The suggested order to implement these in depends on each function's test prerequisites, which you can view in the `test_result.html` file.

**Push** is probably the most important function to get implemented first (that, and its recursive counterpart). That way you can test the other functions as well.

## Recursive functions

Several functions have "front-facing" public functions, and recursive private functions.

- `GetInOrder`
- `GetPreOrder`
- `GetPostOrder`
- `GetMinKey` / `GetMin`
- `GetMaxKey` / `GetMax`
- `GetHeight`
- `Push` / `RecursivePush`
- `FindNode` / `RecursiveFindNode`
- `FindParentOfNode`

## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

BinarySearchTree< TK, TD > . . . . .	9
BinarySearchTree< float, CarData > . . . . .	9
CarData . . . . .	19
CarProgram . . . . .	19
LinkedList< TK, TD > . . . . .	20
LinkedList< float, CarData > . . . . .	20
Node< TK, TD > . . . . .	22
Node< float, CarData > . . . . .	22
TesterBase	
Tester . . . . .	24
Timer . . . . .	25



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">BinarySearchTree&lt; TK, TD &gt;</a>	
A template binary search tree class that takes a KEY and a DATA . . . . .	9
<a href="#">CarData</a> . . . . .	19
<a href="#">CarProgram</a> . . . . .	19
<a href="#">LinkedList&lt; TK, TD &gt;</a> . . . . .	20
<a href="#">Node&lt; TK, TD &gt;</a>	
A template node class, to be used in the <a href="#">BinarySearchTree</a> class . . . . .	22
<a href="#">Tester</a>	
A tester that runs a series of unit tests on the <a href="#">BinarySearchTree</a> object, it will output a <b>test_↔</b> <b>result.txt</b> file with the results . . . . .	24
<a href="#">Timer</a> . . . . .	25



## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

<a href="#">BinarySearchTree.hpp</a>	27
<a href="#">CarData.hpp</a>	27
<a href="#">CarProgram.cpp</a>	27
<a href="#">CarProgram.hpp</a>	28
<a href="#">LinkedList.hpp</a>	28
<a href="#">main.cpp</a>	28
<a href="#">Node.hpp</a>	28
<a href="#">program_main.cpp</a>	29
<a href="#">Tester.hpp</a>	29
<a href="#">Timer.hpp</a>	29





## Chapter 5

# Class Documentation

### 5.1 BinarySearchTree< TK, TD > Class Template Reference

A template binary search tree class that takes a KEY and a DATA.

```
#include <BinarySearchTree.hpp>
```

#### Public Member Functions

- [BinarySearchTree](#) ()  
*Initializes the node count to 0, and the root pointer to nullptr.*
- [~BinarySearchTree](#) ()  
*Destroys the root node.*
- void [Push](#) (const TK &newKey, const TD &newData)  
*Creates a new node for the tree and assigns the data of that node to the parameter passed in.*
- void [Delete](#) (const TK &key)  
*Deletes the [Node](#) that contains the given data, if it exists in the tree.*
- bool [Contains](#) (const TK &key)  
*Searches the tree for the key provided and returns true if found, false if not.*
- bool [RecursiveContains](#) (const TK &key, [Node](#)< TK, TD > \*ptrCurrent)  
*Recursively searches the tree and returns true if found, and false if not.*
- TD \* [GetData](#) (const TK &key)  
*Returns the data based on a key passed in, or nullptr if not present.*
- [Node](#)< TK, TD > \* [FindNode](#) (const TK &key)  
*Returns the Node\* that contains the key, or nullptr if data is not in the tree.*
- string [GetInOrder](#) ()  
*Displays the keys of the nodes in the tree, in in-order format.*
- string [GetPreOrder](#) ()  
*Displays the keys of the nodes in the tree, in pre-order format.*
- string [GetPostOrder](#) ()  
*Displays the keys of the nodes in the tree, in post-order format.*
- TK \* [GetMinKey](#) ()  
*Returns the smallest value in the tree.*
- TK \* [GetMaxKey](#) ()  
*Returns the largest value in the tree.*
- int [GetCount](#) ()  
*Returns the amount of nodes in the tree.*
- int [GetHeight](#) ()  
*Returns the height of the entire tree; must be max height.*

## Private Member Functions

- [Node](#)< TK, TD > \* [RecursiveFindNode](#) (const TK &key, [Node](#)< TK, TD > \*ptrCurrent)
- [Node](#)< TK, TD > \* [FindParentOfNode](#) (const TK &key)  
*Returns the PARENT Node\* of the Node\* that contains the key, or nullptr if data is not in the tree.*
- void [RecursivePush](#) (const TK &newKey, const TD &newData, [Node](#)< TK, TD > \*ptrCurrent)  
*Recurses through the tree and finds the proper location for the new data.*
- void [GetInOrder](#) ([Node](#)< TK, TD > \*ptrCurrent, stringstream &stream)  
*Recurses through the tree in IN-ORDER order, writing to the stream.*
- void [GetPreOrder](#) ([Node](#)< TK, TD > \*ptrCurrent, stringstream &stream)  
*Recurses through the tree in PRE-ORDER order, writing to the stream.*
- void [GetPostOrder](#) ([Node](#)< TK, TD > \*ptrCurrent, stringstream &stream)  
*Recurses through the tree in POST-ORDER order, writing to the stream.*
- TK \* [GetMax](#) ([Node](#)< TK, TD > \*ptrCurrent)  
*Recurses through the tree, going to the right-child-nodes until the max key is found.*
- TK \* [GetMin](#) ([Node](#)< TK, TD > \*ptrCurrent)  
*Recurses through the tree, going to the right-child-nodes until the max key is found.*
- int [GetHeight](#) ([Node](#)< TK, TD > \*ptrCurrent)  
*Get the height of the tree (the longest path from the root to the lowest leaf)*
- void [DeleteNode\\_NoChildren](#) ([Node](#)< TK, TD > \*deleteMe, [Node](#)< TK, TD > \*parent, bool isLeftNode)  
*Deletes the given node, which has no children.*
- void [DeleteNode\\_LeftChild](#) ([Node](#)< TK, TD > \*deleteMe, [Node](#)< TK, TD > \*parent, bool isLeftNode)  
*Deletes the given node, which has a left child but no right child.*
- void [DeleteNode\\_RightChild](#) ([Node](#)< TK, TD > \*deleteMe, [Node](#)< TK, TD > \*parent, bool isLeftNode)  
*Deletes the given node, which has a left child but no right.*
- void [DeleteNode\\_TwoChildren](#) ([Node](#)< TK, TD > \*deleteMe, [Node](#)< TK, TD > \*parent, bool isLeftNode)  
*Deletes the given node, which has children to the left and to the right.*

## Private Attributes

- [Node](#)< TK, TD > \* [m\\_ptrRoot](#)  
*A pointer to the root node of the tree; TK = data type of the key, TD = data type of the data.*
- int [m\\_nodeCount](#)  
*The amount of nodes in the tree.*

## Friends

- class [Tester](#)

### 5.1.1 Detailed Description

```
template<typename TK, typename TD>
class BinarySearchTree< TK, TD >
```

A template binary search tree class that takes a KEY and a DATA.

### 5.1.2 Constructor & Destructor Documentation

#### 5.1.2.1 `template<typename TK, typename TD> BinarySearchTree< TK, TD >::BinarySearchTree ( )`

Initializes the node count to 0, and the root pointer to nullptr.

#### 5.1.2.2 `template<typename TK, typename TD> BinarySearchTree< TK, TD >::~~BinarySearchTree ( )`

Destroys the root node.

### 5.1.3 Member Function Documentation

#### 5.1.3.1 `template<typename TK, typename TD> bool BinarySearchTree< TK, TD >::Contains ( const TK & key )`

Searches the tree for the key provided and returns true if found, false if not.

##### Returns

<bool> whether or not the key is found in the tree.

#### 5.1.3.2 `template<typename TK, typename TD> void BinarySearchTree< TK, TD >::Delete ( const TK & key )`

Deletes the [Node](#) that contains the given data, if it exists in the tree.

Search the tree for a [Node](#) that contains the data. This [Node](#) will be removed. BEFORE you remove it, you need to relocate its children AND set the "deleteMe" node's children to nullptr.

- Create a [Node](#) pointer called deleteMe. Use FindNode to find the node to delete via its key.
- Create a [Node](#) pointer called parent. Use the FindParentOfNode with the key to find its parent.
- Create a boolean called isLeftNode. If the parent's left child is the deleteMe node, set this to true. Otherwise, false.
- If the deleteMe node is nullptr, then return.
- If the deleteMe pointer has no children, call DeleteNode\_NoChildren
- If the deleteMe pointer has only a left child, call DeleteNode\_LeftChild
- If the deleteMe pointer has only a right child call DeleteNode\_RightChild
- If the deleteMe pointer has a left AND a right child, call DeleteNode\_TwoChildren

##### Parameters

<i>data</i>	const TK&, the data to be removed from the tree.
-------------	--

## Returns

&lt;void&gt;

5.1.3.3 `template<typename TK, typename TD> void BinarySearchTree< TK, TD >::DeleteNode_LeftChild ( Node< TK, TD > * deleteMe, Node< TK, TD > * parent, bool isLeftNode ) [private]`

Deletes the given node, which has a left child but no right child.

- If deleteMe is the root: Set m\_ptrRoot to the deleteMe's ptrLeft.
- If deleteMe is the left node: Set the parent's ptrLeft to deleteMe's ptrLeft.
- If deleteMe is the right node: Set the parent's ptrRight to deleteMe's ptrLeft.
- Set deleteMe's left ptr to nullptr.
- Set deleteMe's right ptr to nullptr.
- Delete deleteMe.
- Decrement the node count.

5.1.3.4 `template<typename TK, typename TD> void BinarySearchTree< TK, TD >::DeleteNode_NoChildren ( Node< TK, TD > * deleteMe, Node< TK, TD > * parent, bool isLeftNode ) [private]`

Deletes the given node, which has no children.

- If deleteMe is the root: Set the m\_ptrRoot to nullptr.
- Else if deleteMe is the left node: Set the parent's ptrLeft to nullptr.
- Else if deleteMe is the right node: Set the parent's ptrRight to nullptr.
- Set deleteMe's left ptr to nullptr.
- Set deleteMe's right ptr to nullptr.
- Delete deleteMe.
- Decrement the node count.

5.1.3.5 `template<typename TK, typename TD> void BinarySearchTree< TK, TD >::DeleteNode_RightChild ( Node< TK, TD > * deleteMe, Node< TK, TD > * parent, bool isLeftNode ) [private]`

Deletes the given node, which has a left child but no right.

- If deleteMe is the root: Set m\_ptrRoot to the deleteMe's ptrRight.
- If deleteMe is the left node: Set the parent's ptrLeft to deleteMe's ptrRight.
- If deleteMe is the right node: Set the parent's ptrRight to deleteMe's ptrRight.
- Set deleteMe's left ptr to nullptr.
- Set deleteMe's right ptr to nullptr.
- Delete deleteMe.
- Decrement the node count.

5.1.3.6 `template<typename TK, typename TD> void BinarySearchTree< TK, TD >::DeleteNode_TwoChildren ( Node< TK, TD > * deleteMe, Node< TK, TD > * parent, bool isLeftNode ) [private]`

Deletes the given node, which has children to the left and to the right.

- Create a new [Node](#) pointer called tempNode, set it to deleteMe's right child.
- Create a new [Node](#) pointer called successor, set it to deleteMe.
- Create a new [Node](#) pointer called successorParent, set it to deleteMe.
- while the tempNode is not equal to nullptr...
  - Set successorParent to the successor
  - Set successor to the tempNode
  - Set tempNode to the tempNode's left child.
- If successor is not equal to deleteMe's right child:
  - Set successorParent's left child to successor's right child.
  - Set successor's right child to deleteMe's right child.
- If deleteMe is the root: Set the root to the successor.
- Else if deleteMe is the left node:
  - Set deleteMe to the parent's ptrLeft
  - Set the parent's ptrLeft to the successor
- Else if deleteMe is the right node:
  - Set deleteMe to the parent's ptrRight
  - Set the parent's ptrRight to the successor
- Set the successor's ptrLeft to deleteMe's ptrLeft.
- Set deleteMe's left ptr to nullptr.
- Set deleteMe's right ptr to nullptr.
- Delete deleteMe.
- Decrement the node count.

5.1.3.7 `template<typename TK, typename TD > Node< TK, TD > * BinarySearchTree< TK, TD >::FindNode ( const TK & key )`

Returns the Node\* that contains the key, or nullptr if data is not in the tree.

Similar to the Contains function, you will have to traverse the tree to find the key. If you find the [Node](#) that contains this data, you will return the Node\*.

#### Parameters

<i>data</i>	<const TK&>, the key that we are searching for.
-------------	---

**Returns**

<Node<T>\*> the Node\* that contains the key. Otherwise, nullptr if data is not found.

**5.1.3.8** `template<typename TK, typename TD > Node< TK, TD > * BinarySearchTree< TK, TD >::FindParentOfNode ( const TK & key ) [private]`

Returns the PARENT Node\* of the Node\* that contains the key, or nullptr if data is not in the tree.

Similar to the FindNode function, you will instead return the PARENT NODE of the node that contains the key, rather than the node itself.

**Parameters**

<i>data</i>	<const TK&>, the key we are searching for; but going to return the parent of this node.
-------------	---

**Returns**

<Node<TK, TD>\*> the Node\* that contains the data. Otherwise, nullptr if key is not found.

**5.1.3.9** `template<typename TK , typename TD > int BinarySearchTree< TK, TD >::GetCount ( )`

Returns the amount of nodes in the tree.

**Returns**

<int> the amount of nodes in the tree

**5.1.3.10** `template<typename TK, typename TD > TD * BinarySearchTree< TK, TD >::GetData ( const TK & key )`

Returns the data based on a key passed in, or nullptr if not present.

Search the binary tree for the node that has the given key (can use the FindNode function), and returns the data associated with that node. If no node is found, return nullptr instead.

**Parameters**

<i>key</i>	<TK&> the key of the node we're searching for
------------	---

**Returns**

<TD\*> data of the node that has the key given

**5.1.3.11** `template<typename TK , typename TD > int BinarySearchTree< TK, TD >::GetHeight ( )`

Returns the height of the entire tree; must be max height.

**Returns**

<int> height of the tree

**5.1.3.12** `template<typename TK, typename TD> int BinarySearchTree< TK, TD >::GetHeight ( Node< TK, TD > * ptrCurrent ) [private]`

Get the height of the tree (the longest path from the root to the lowest leaf)

Get the height of left sub tree, say leftHeight Get the height of right sub tree, say rightHeight Take the Max(leftHeight, rightHeight) and add 1 for the root and return Call recursively.

**5.1.3.13** `template<typename TK , typename TD > string BinarySearchTree< TK, TD >::GetInOrder ( )`

Displays the keys of the nodes in the tree, in in-order format.

This function creates a stringstream and calls the recursive GetInOrder function. It will return the keys of the nodes in the tree in string format.

**Returns**

<string> The keys of the nodes in the tree, in in-order format, as a string.

**5.1.3.14** `template<typename TK, typename TD> void BinarySearchTree< TK, TD >::GetInOrder ( Node< TK, TD > * ptrCurrent, stringstream & stream ) [private]`

Recurses through the tree in IN-ORDER order, writing to the stream.

In order will display the items in the tree in ascending order. From an algorithmic point of view, for whatever node it is on, it will:

- Display the left node item (recurse GetInOrder on left child)
- Display the current node item
- Display the right node item (recurse GetInOrder on right child)

To write out to the stream, simply use: `stream << ptrCurrent->key << " "`;

Note that it is expected that you will have a leading space " " at the end of the generated string.

**5.1.3.15** `template<typename TK, typename TD> TK * BinarySearchTree< TK, TD >::GetMax ( Node< TK, TD > * ptrCurrent ) [private]`

Recurses through the tree, going to the right-child-nodes until the max key is found.

Items are inserted into a binary search tree in a sorted order. This means that larger items always get inserted to the right, with the maximum value being the right-most value.

**5.1.3.16** `template<typename TK, typename TD> TK * BinarySearchTree< TK, TD >::GetMaxKey ( )`

Returns the largest value in the tree.

Keep in mind that, for any given node, anything to the RIGHT is greater than that node.

**Returns**

<TK\*> Largest key in the tree, or nullptr if tree is empty

**5.1.3.17** `template<typename TK, typename TD> TK * BinarySearchTree< TK, TD >::GetMin ( Node< TK, TD > * ptrCurrent ) [private]`

Recurses through the tree, going to the right-child-nodes until the max key is found.

Items are inserted into a binary search tree in a sorted order. This means that larger items always get inserted to the left, with the minimum value being the left-most value.

**5.1.3.18** `template<typename TK, typename TD> TK * BinarySearchTree< TK, TD >::GetMinKey ( )`

Returns the smallest value in the tree.

Keep in mind that, for any given node, anything to the LEFT is greater than that node.

**Returns**

<TK\*> Smallest key in the tree, or nullptr if tree is empty

**5.1.3.19** `template<typename TK, typename TD> string BinarySearchTree< TK, TD >::GetPostOrder ( )`

Displays the keys of the nodes in the tree, in post-order format.

This function creates a stringstream and calls the recursive GetPostOrder function. It will return the keys of the nodes in the tree in string format.

**Returns**

<string> The keys of the nodes in the tree, in post-order format, as a string.

**5.1.3.20** `template<typename TK, typename TD> void BinarySearchTree< TK, TD >::GetPostOrder ( Node< TK, TD > * ptrCurrent, stringstream & stream ) [private]`

Recurses through the tree in POST-ORDER order, writing to the stream.

In order will display the items in the tree post-order. From an algorithmic point of view, for whatever node it is on, it will:

- Display the left node item
- Display the right node item
- Display the current node item

Note that it is expected that you will have a leading space " " at the end of the generated string.



### 5.1.3.21 `template<typename TK, typename TD> string BinarySearchTree< TK, TD >::GetPreOrder ( )`

Displays the keys of the nodes in the tree, in pre-order format.

This function creates a stringstream and calls the recursive GetPreOrder function. It will return the keys of the nodes in the tree in string format.

#### Returns

<string> The keys of the nodes in the tree, in pre-order format, as a string.

### 5.1.3.22 `template<typename TK, typename TD> void BinarySearchTree< TK, TD >::GetPreOrder ( Node< TK, TD > * ptrCurrent, stringstream & stream ) [private]`

Recurse through the tree in PRE-ORDER order, writing to the stream.

In order will display the items in the tree pre-order. From an algorithmic point of view, for whatever node it is on, it will:

- Display the current node item
- Display the left node item
- Display the right node item

Note that it is expected that you will have a leading space " " at the end of the generated string.

### 5.1.3.23 `template<typename TK, typename TD> void BinarySearchTree< TK, TD >::Push ( const TK & newKey, const TD & newData )`

Creates a new node for the tree and assigns the data of that node to the parameter passed in.

If the tree is empty, the new data goes at the root node. Otherwise, we will have to recurse through the tree in order to find the correct position for our new data.

Step-by-step:

- if root is null...
  - Create a new node via the root pointer. Set its key and data.
  - Increment the node count
- else...
  - Call RecursivePush with the newKey, newData, and the root.

#### Parameters

<i>newKey</i>	const TK&, the key to index this new item under
<i>newData</i>	const TD&, the new data to be added to the tree

**Returns**

&lt;void&gt;

5.1.3.24 `template<typename TK, typename TD> bool BinarySearchTree< TK, TD >::RecursiveContains ( const TK & key, Node< TK, TD > * ptrCurrent )`

Recursively searches the tree and returns true if found, and false if not.

TERMINATING CASE 1: If the ptrCurrent is nullptr, return false.

TERMINATING CASE 2: If the ptrCurrent's key is equal to the key, return true.

RECURSIVE CASE: Call both RecursiveContains( key, ptrCurrent->ptrLeft ) and RecursiveContains( key, ptrCurrent->ptrRight ). If either of these return true, then return true.

5.1.3.25 `template<typename TK, typename TD> Node< TK, TD > * BinarySearchTree< TK, TD >::RecursiveFindNode ( const TK & key, Node< TK, TD > * ptrCurrent ) [private]`

5.1.3.26 `template<typename TK, typename TD> void BinarySearchTree< TK, TD >::RecursivePush ( const TK & newKey, const TD & newData, Node< TK, TD > * ptrCurrent ) [private]`

Recurses through the tree and finds the proper location for the new data.

TERMINATING CASE: If ptrCurrent is a nullptr, then we have traversed to a space that is available for our new node. Create our new node here and set up the data.

RECURSIVE CASE: If ptrCurrent is already taken, we must figure out whether to recurse LEFT or RIGHT, based on the value of newData compared to the ptrCurrent's data.

**Parameters**

<i>newKey</i>	const TK&, the key to index this new item under
<i>newData</i>	const TD&, the new data to be added to the tree
<i>ptrCurrent</i>	Node<TK, TD>*, the pointer to the current root of a subtree; will traverse left or right if newKey is less-than or greater-than, respectively.

**Returns**

&lt;void&gt;

**5.1.4 Friends And Related Function Documentation**

5.1.4.1 `template<typename TK, typename TD> friend class Tester [friend]`

**5.1.5 Member Data Documentation**

5.1.5.1 `template<typename TK, typename TD> int BinarySearchTree< TK, TD >::m_nodeCount [private]`

The amount of nodes in the tree.

5.1.5.2 `template<typename TK, typename TD> Node<TK, TD>* BinarySearchTree< TK, TD >::m_ptrRoot`  
`[private]`

A pointer to the root node of the tree; TK = data type of the key, TD = data type of the data.

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

- [BinarySearchTree.hpp](#)

## 5.2 CarData Struct Reference

```
#include <CarData.hpp>
```

### Public Attributes

- string [make](#)
- string [model](#)
- float [price](#)

### 5.2.1 Member Data Documentation

5.2.1.1 string CarData::make

5.2.1.2 string CarData::model

5.2.1.3 float CarData::price

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

- [CarData.hpp](#)

## 5.3 CarProgram Class Reference

```
#include <CarProgram.hpp>
```

Collaboration diagram for CarProgram:

### Public Member Functions

- void [Start](#) ()
- void [MainLoop](#) ()
- void [PrintStats](#) (const string &outFile, const string &orderFile)

## Private Member Functions

- void [LoadData](#) (const string &carFile)
- void [SaveData](#) (const string &carFile)

## Private Attributes

- [BinarySearchTree](#)< float, [CarData](#) > [m\\_data\\_tree](#)
- [LinkedList](#)< float, [CarData](#) > [m\\_data\\_linear](#)

### 5.3.1 Member Function Documentation

5.3.1.1 void [CarProgram::LoadData](#) ( const string & *carFile* ) [private]

5.3.1.2 void [CarProgram::MainLoop](#) ( )

5.3.1.3 void [CarProgram::PrintStats](#) ( const string & *outFile*, const string & *orderFile* )

5.3.1.4 void [CarProgram::SaveData](#) ( const string & *carFile* ) [private]

5.3.1.5 void [CarProgram::Start](#) ( )

### 5.3.2 Member Data Documentation

5.3.2.1 [LinkedList](#)<float, [CarData](#)> [CarProgram::m\\_data\\_linear](#) [private]

5.3.2.2 [BinarySearchTree](#)<float, [CarData](#)> [CarProgram::m\\_data\\_tree](#) [private]

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

- [CarProgram.hpp](#)
- [CarProgram.cpp](#)

## 5.4 [LinkedList](#)< TK, TD > Class Template Reference

```
#include <LinkedList.hpp>
```

## Public Member Functions

- [LinkedList](#) ()
- [~LinkedList](#) ()
- void [Clear](#) ()
- void [PushBack](#) (const TK &newKey, const TD &newData)
- void [PushFront](#) (const TK &newKey, const TD &newData)
- void [PopBack](#) ()
- void [PopFront](#) ()
- TD & [GetFirst](#) ()
- TD & [GetLast](#) ()
- TD & [GetItemAtPosition](#) (int index)
- TD \* [GetItemWithKey](#) (const TK &key)
- bool [Contains](#) (TD item)
- int [Size](#) ()
- bool [IsEmpty](#) ()
- void [Display](#) ()

## Private Attributes

- [Node](#)< TK, TD > \* [ptrFront](#)
- [Node](#)< TK, TD > \* [ptrEnd](#)
- int [m\\_size](#)

### 5.4.1 Constructor & Destructor Documentation

5.4.1.1 `template<typename TK, typename TD> LinkedList< TK, TD >::LinkedList ( )` `[inline]`

5.4.1.2 `template<typename TK, typename TD> LinkedList< TK, TD >::~~LinkedList ( )` `[inline]`

### 5.4.2 Member Function Documentation

5.4.2.1 `template<typename TK, typename TD> void LinkedList< TK, TD >::Clear ( )` `[inline]`

5.4.2.2 `template<typename TK, typename TD> bool LinkedList< TK, TD >::Contains ( TD item )` `[inline]`

5.4.2.3 `template<typename TK, typename TD> void LinkedList< TK, TD >::Display ( )` `[inline]`

5.4.2.4 `template<typename TK, typename TD> TD& LinkedList< TK, TD >::GetFirst ( )` `[inline]`

5.4.2.5 `template<typename TK, typename TD> TD& LinkedList< TK, TD >::GetItemAtPosition ( int index )` `[inline]`

5.4.2.6 `template<typename TK, typename TD> TD* LinkedList< TK, TD >::GetItemWithKey ( const TK & key )`  
`[inline]`

5.4.2.7 `template<typename TK, typename TD> TD& LinkedList< TK, TD >::GetLast ( )` `[inline]`

5.4.2.8 `template<typename TK, typename TD> bool LinkedList< TK, TD >::IsEmpty ( )` `[inline]`

5.4.2.9 `template<typename TK, typename TD> void LinkedList< TK, TD >::PopBack ( ) [inline]`

5.4.2.10 `template<typename TK, typename TD> void LinkedList< TK, TD >::PopFront ( ) [inline]`

5.4.2.11 `template<typename TK, typename TD> void LinkedList< TK, TD >::PushBack ( const TK & newKey, const TD & newData ) [inline]`

5.4.2.12 `template<typename TK, typename TD> void LinkedList< TK, TD >::PushFront ( const TK & newKey, const TD & newData ) [inline]`

5.4.2.13 `template<typename TK, typename TD> int LinkedList< TK, TD >::Size ( ) [inline]`

### 5.4.3 Member Data Documentation

5.4.3.1 `template<typename TK, typename TD> int LinkedList< TK, TD >::m_size [private]`

5.4.3.2 `template<typename TK, typename TD> Node<TK, TD>* LinkedList< TK, TD >::ptrEnd [private]`

5.4.3.3 `template<typename TK, typename TD> Node<TK, TD>* LinkedList< TK, TD >::ptrFront [private]`

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

- [LinkedList.hpp](#)

## 5.5 Node< TK, TD > Class Template Reference

A template node class, to be used in the [BinarySearchTree](#) class.

```
#include <Node.hpp>
```

Collaboration diagram for Node< TK, TD >:

### Public Member Functions

- [Node](#) ()  
*Initializes left and right pointers to nullptr.*
- [Node](#) (TK newKey, TD newData)
- void [IsNotTree](#) ()
- [~Node](#) ()  
*Destroys left and right children, if they are not pointing to nullptr.*

## Public Attributes

- [Node< TK, TD > \\* ptrLeft](#)  
*Pointer to the left child of the node, which is lesser in value.*
- [Node< TK, TD > \\* ptrRight](#)  
*Pointer to the right child of the node, which is greater in value.*
- TD [data](#)  
*The data stored by the node.*
- TK [key](#)  
*The key of this node.*
- bool [isTree](#)

### 5.5.1 Detailed Description

```
template<typename TK, typename TD>
class Node< TK, TD >
```

A template node class, to be used in the [BinarySearchTree](#) class.

### 5.5.2 Constructor & Destructor Documentation

5.5.2.1 `template<typename TK, typename TD> Node< TK, TD >::Node ( ) [inline]`

Initializes left and right pointers to nullptr.

5.5.2.2 `template<typename TK, typename TD> Node< TK, TD >::Node ( TK newKey, TD newData ) [inline]`

5.5.2.3 `template<typename TK, typename TD> Node< TK, TD >::~~Node ( ) [inline]`

Destroys left and right children, if they are not pointing to nullptr.

### 5.5.3 Member Function Documentation

5.5.3.1 `template<typename TK, typename TD> void Node< TK, TD >::isNotTree ( ) [inline]`

### 5.5.4 Member Data Documentation

5.5.4.1 `template<typename TK, typename TD> TD Node< TK, TD >::data`

The data stored by the node.

5.5.4.2 `template<typename TK, typename TD> bool Node< TK, TD >::isTree`

5.5.4.3 `template<typename TK, typename TD> TK Node< TK, TD >::key`

The key of this node.

5.5.4.4 `template<typename TK, typename TD> Node<TK, TD>* Node< TK, TD >::ptrLeft`

Pointer to the left child of the node, which is lesser in value.

5.5.4.5 `template<typename TK, typename TD> Node<TK, TD>* Node< TK, TD >::ptrRight`

Pointer to the right child of the node, which is greater in value.

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

- [Node.hpp](#)

## 5.6 Tester Class Reference

A tester that runs a series of unit tests on the [BinarySearchTree](#) object, it will output a **test\_result.txt** file with the results.

```
#include <Tester.hpp>
```

Inheritance diagram for Tester:

Collaboration diagram for Tester:

### Public Member Functions

- [Tester \(\)](#)

### Private Member Functions

- int [Test\\_Push \(\)](#)
- int [Test\\_Delete \(\)](#)
- int [Test\\_Contains \(\)](#)
- int [Test\\_FindNode \(\)](#)
- int [Test\\_FindParentOfNode \(\)](#)
- int [Test\\_GetInOrder \(\)](#)
- int [Test\\_GetPreOrder \(\)](#)
- int [Test\\_GetPostOrder \(\)](#)
- int [Test\\_GetMax \(\)](#)
- int [Test\\_GetCount \(\)](#)
- int [Test\\_GetHeight \(\)](#)

#### 5.6.1 Detailed Description

A tester that runs a series of unit tests on the [BinarySearchTree](#) object, it will output a **test\_result.txt** file with the results.



## 5.6.2 Constructor & Destructor Documentation

5.6.2.1 `Tester::Tester ( )` `[inline]`

## 5.6.3 Member Function Documentation

5.6.3.1 `int Tester::Test_Contains ( )` `[private]`

5.6.3.2 `int Tester::Test_Delete ( )` `[private]`

5.6.3.3 `int Tester::Test_FindNode ( )` `[private]`

5.6.3.4 `int Tester::Test_FindParentOfNode ( )` `[private]`

5.6.3.5 `int Tester::Test_GetCount ( )` `[private]`

5.6.3.6 `int Tester::Test_GetHeight ( )` `[private]`

5.6.3.7 `int Tester::Test_GetInOrder ( )` `[private]`

5.6.3.8 `int Tester::Test_GetMax ( )` `[private]`

5.6.3.9 `int Tester::Test_GetPostOrder ( )` `[private]`

5.6.3.10 `int Tester::Test_GetPreOrder ( )` `[private]`

5.6.3.11 `int Tester::Test_Push ( )` `[private]`

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

- [Tester.hpp](#)

## 5.7 Timer Class Reference

```
#include <Timer.hpp>
```

### Public Member Functions

- void [Start](#) ()
- unsigned int [GetElapsedSeconds](#) ()
- unsigned int [GetElapsedMilliseconds](#) ()

### Private Attributes

- chrono::system\_clock::time\_point [m\\_startTime](#)

### 5.7.1 Member Function Documentation

5.7.1.1 `unsigned int Timer::GetElapsedMilliseconds ( ) [inline]`

5.7.1.2 `unsigned int Timer::GetElapsedSeconds ( ) [inline]`

5.7.1.3 `void Timer::Start ( ) [inline]`

### 5.7.2 Member Data Documentation

5.7.2.1 `chrono::system_clock::time_point Timer::m_startTime [private]`

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

- [Timer.hpp](#)

## Chapter 6

# File Documentation

### 6.1 BinarySearchTree.hpp File Reference

```
#include <iostream>
#include <string>
#include <sstream>
#include "Node.hpp"
Include dependency graph for BinarySearchTree.hpp:
```

### 6.2 car-data.txt File Reference

### 6.3 CarData.hpp File Reference

```
#include <string>
Include dependency graph for CarData.hpp: This graph shows which files directly or indirectly include this file:
```

#### Classes

- struct [CarData](#)

### 6.4 CarProgram.cpp File Reference

```
#include "CarProgram.hpp"
#include <fstream>
#include <iomanip>
#include "cuTEST/StringUtil.hpp"
#include "Timer.hpp"
Include dependency graph for CarProgram.cpp:
```

## 6.5 CarProgram.hpp File Reference

```
#include "cuTEST/Menu.hpp"  
#include "CarData.hpp"  
#include "BinarySearchTree.hpp"  
#include "LinkedList.hpp"
```

Include dependency graph for CarProgram.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [CarProgram](#)

## 6.6 data-car-makes.txt File Reference

## 6.7 LinkedList.hpp File Reference

```
#include <iostream>  
#include "Node.hpp"
```

Include dependency graph for LinkedList.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- class [LinkedList](#)< TK, TD >

## 6.8 main.cpp File Reference

```
#include <iostream>  
#include "Tester.hpp"  
#include "CarProgram.hpp"  
#include "cuTEST/Menu.hpp"
```

Include dependency graph for main.cpp:

### Functions

- int [main](#) ()

### 6.8.1 Function Documentation

#### 6.8.1.1 int main ( )

## 6.9 main.md File Reference

## 6.10 Node.hpp File Reference

This graph shows which files directly or indirectly include this file:

## Classes

- class [Node< TK, TD >](#)  
A template node class, to be used in the [BinarySearchTree](#) class.

## 6.11 program\_main.cpp File Reference

```
#include "EmployeeManager.hpp"
Include dependency graph for program_main.cpp:
```

## Functions

- int [main](#) ()

### 6.11.1 Function Documentation

#### 6.11.1.1 int main ( )

## 6.12 Tester.hpp File Reference

```
#include <iostream>
#include <string>
#include "cuTEST/TesterBase.hpp"
#include "cuTEST/Menu.hpp"
#include "cuTEST/StringUtil.hpp"
#include "BinarySearchTree.hpp"
```

Include dependency graph for Tester.hpp: This graph shows which files directly or indirectly include this file:

## Classes

- class [Tester](#)  
A tester that runs a series of unit tests on the [BinarySearchTree](#) object, it will output a **test\_result.txt** file with the results.

## Macros

- [#define \\_TESTER\\_HPP](#)

### 6.12.1 Macro Definition Documentation

#### 6.12.1.1 [#define \\_TESTER\\_HPP](#)

## 6.13 Timer.hpp File Reference

```
#include <chrono>
Include dependency graph for Timer.hpp: This graph shows which files directly or indirectly include this file:
```

## Classes

- class [Timer](#)



# Index

- `_TESTER_HPP`
    - `Tester.hpp`, 29
  - `~BinarySearchTree`
    - `BinarySearchTree`, 11
  - `~LinkedList`
    - `LinkedList`, 21
  - `~Node`
    - `Node`, 23
- `BinarySearchTree`
  - `~BinarySearchTree`, 11
  - `BinarySearchTree`, 11
  - `Contains`, 11
  - `Delete`, 11
  - `DeleteNode_LeftChild`, 12
  - `DeleteNode_NoChildren`, 12
  - `DeleteNode_RightChild`, 12
  - `DeleteNode_TwoChildren`, 12
  - `FindNode`, 13
  - `FindParentOfNode`, 14
  - `GetCount`, 14
  - `GetData`, 14
  - `GetHeight`, 14, 15
  - `GetInOrder`, 15
  - `GetMax`, 15
  - `GetMaxKey`, 15
  - `GetMin`, 16
  - `GetMinKey`, 16
  - `GetPostOrder`, 16
  - `GetPreOrder`, 16, 17
  - `m_nodeCount`, 18
  - `m_ptrRoot`, 18
  - `Push`, 17
  - `RecursiveContains`, 18
  - `RecursiveFindNode`, 18
  - `RecursivePush`, 18
  - `Tester`, 18
- `BinarySearchTree< TK, TD >`, 9
- `BinarySearchTree.hpp`, 27
- `car-data.txt`, 27
- `CarData`, 19
  - `make`, 19
  - `model`, 19
  - `price`, 19
- `CarData.hpp`, 27
- `CarProgram`, 19
  - `LoadData`, 20
  - `m_data_linear`, 20
  - `m_data_tree`, 20
  - `MainLoop`, 20
  - `PrintStats`, 20
  - `SaveData`, 20
  - `Start`, 20
- `CarProgram.cpp`, 27
- `CarProgram.hpp`, 28
- `Clear`
  - `LinkedList`, 21
- `Contains`
  - `BinarySearchTree`, 11
  - `LinkedList`, 21
- `data`
  - `Node`, 23
- `data-car-makes.txt`, 28
- `Delete`
  - `BinarySearchTree`, 11
- `DeleteNode_LeftChild`
  - `BinarySearchTree`, 12
- `DeleteNode_NoChildren`
  - `BinarySearchTree`, 12
- `DeleteNode_RightChild`
  - `BinarySearchTree`, 12
- `DeleteNode_TwoChildren`
  - `BinarySearchTree`, 12
- `Display`
  - `LinkedList`, 21
- `FindNode`
  - `BinarySearchTree`, 13
- `FindParentOfNode`
  - `BinarySearchTree`, 14
- `GetCount`
  - `BinarySearchTree`, 14
- `GetData`
  - `BinarySearchTree`, 14
- `GetElapsedMilliseconds`
  - `Timer`, 26
- `GetElapsedSeconds`
  - `Timer`, 26
- `GetFirst`
  - `LinkedList`, 21
- `GetHeight`
  - `BinarySearchTree`, 14, 15
- `GetInOrder`
  - `BinarySearchTree`, 15
- `GetItemAtPosition`
  - `LinkedList`, 21
- `GetItemWithKey`

- LinkedList, 21
- GetLast
  - LinkedList, 21
- GetMax
  - BinarySearchTree, 15
- GetMaxKey
  - BinarySearchTree, 15
- GetMin
  - BinarySearchTree, 16
- GetMinKey
  - BinarySearchTree, 16
- GetPostOrder
  - BinarySearchTree, 16
- GetPreOrder
  - BinarySearchTree, 16, 17
- IsEmpty
  - LinkedList, 21
- IsNotTree
  - Node, 23
- isTree
  - Node, 23
- key
  - Node, 23
- LinkedList
  - ~LinkedList, 21
  - Clear, 21
  - Contains, 21
  - Display, 21
  - GetFirst, 21
  - GetItemAtPosition, 21
  - GetItemWithKey, 21
  - GetLast, 21
  - IsEmpty, 21
  - LinkedList, 21
  - m\_size, 22
  - PopBack, 21
  - PopFront, 22
  - ptrEnd, 22
  - ptrFront, 22
  - PushBack, 22
  - PushFront, 22
  - Size, 22
- LinkedList< TK, TD >, 20
- LinkedList.hpp, 28
- LoadData
  - CarProgram, 20
- m\_data\_linear
  - CarProgram, 20
- m\_data\_tree
  - CarProgram, 20
- m\_nodeCount
  - BinarySearchTree, 18
- m\_ptrRoot
  - BinarySearchTree, 18
- m\_size
  - LinkedList, 22
- m\_startTime
  - Timer, 26
- main
  - main.cpp, 28
  - program\_main.cpp, 29
- main.cpp, 28
  - main, 28
- main.md, 28
- MainLoop
  - CarProgram, 20
- make
  - CarData, 19
- model
  - CarData, 19
- Node
  - ~Node, 23
  - data, 23
  - IsNotTree, 23
  - isTree, 23
  - key, 23
  - Node, 23
  - ptrLeft, 23
  - ptrRight, 24
- Node< TK, TD >, 22
- Node.hpp, 28
- PopBack
  - LinkedList, 21
- PopFront
  - LinkedList, 22
- price
  - CarData, 19
- PrintStats
  - CarProgram, 20
- program\_main.cpp, 29
  - main, 29
- ptrEnd
  - LinkedList, 22
- ptrFront
  - LinkedList, 22
- ptrLeft
  - Node, 23
- ptrRight
  - Node, 24
- Push
  - BinarySearchTree, 17
- PushBack
  - LinkedList, 22
- PushFront
  - LinkedList, 22
- RecursiveContains
  - BinarySearchTree, 18
- RecursiveFindNode
  - BinarySearchTree, 18
- RecursivePush
  - BinarySearchTree, 18



- SaveData
  - CarProgram, [20](#)
- Size
  - LinkedList, [22](#)
- Start
  - CarProgram, [20](#)
  - Timer, [26](#)
- Test\_Contains
  - Tester, [25](#)
- Test\_Delete
  - Tester, [25](#)
- Test\_FindNode
  - Tester, [25](#)
- Test\_FindParentOfNode
  - Tester, [25](#)
- Test\_GetCount
  - Tester, [25](#)
- Test\_GetHeight
  - Tester, [25](#)
- Test\_GetInOrder
  - Tester, [25](#)
- Test\_GetMax
  - Tester, [25](#)
- Test\_GetPostOrder
  - Tester, [25](#)
- Test\_GetPreOrder
  - Tester, [25](#)
- Test\_Push
  - Tester, [25](#)
- Tester, [24](#)
  - BinarySearchTree, [18](#)
  - Test\_Contains, [25](#)
  - Test\_Delete, [25](#)
  - Test\_FindNode, [25](#)
  - Test\_FindParentOfNode, [25](#)
  - Test\_GetCount, [25](#)
  - Test\_GetHeight, [25](#)
  - Test\_GetInOrder, [25](#)
  - Test\_GetMax, [25](#)
  - Test\_GetPostOrder, [25](#)
  - Test\_GetPreOrder, [25](#)
  - Test\_Push, [25](#)
  - Tester, [25](#)
- Tester.hpp, [29](#)
  - \_TESTER\_HPP, [29](#)
- Timer, [25](#)
  - GetElapsedMilliseconds, [26](#)
  - GetElapsedSeconds, [26](#)
  - m\_startTime, [26](#)
  - Start, [26](#)
- Timer.hpp, [29](#)