PA08-HashTable

Generated by Doxygen 1.8.11

Contents

1 Main Page

This program contains the necessary functions to implement -a Binary Search Tree ADT using a linked tree structure

While doing this project I had to keep in mind that I was using the Binary Search Tree to create the dataTable of the HashTable. I had to use my Binary Search Tree functions to implement the HashTable entirely.

2 Class Index

2.1 Class List

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

3 File Index

3.1 File List

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

4 Class Documentation

4.1 BSTree< DataType, KeyType > Class Template Reference

Classes

- class BSTreeNode

Public Member Functions

- BSTree ()
- BSTree (const BSTree< DataType, KeyType > &other)
- BSTree & operator= (const BSTree< DataType, KeyType > &other)
- ~BSTree ()
- void insert (const DataType &newDataItem)
- bool retrieve (const KeyType &searchKey, DataType &searchDataItem) const
- bool remove (const KeyType &deleteKey)
- void writeKeys () const
- void clear ()
- bool isEmpty () const
- void showStructure () const
- int getHeight () const
- int getCount () const
- void **writeLessThan** (const KeyType &searchKey) const

Protected Member Functions

- void showHelper (BSTreeNode *p, int level) const
- void copyHelper (BSTreeNode *&dest, BSTreeNode *src)
- void insertHelper (BSTreeNode *&current, const DataType &newDataItem)
- bool retrieveHelper (BSTreeNode *current, const KeyType &searchKey, DataType &searchDataItem) const
- bool removeHelper (BSTreeNode *&current, const KeyType &deleteKey)
- void writeKeysHelper (const BSTreeNode *current) const
- void clearHelper (BSTreeNode *&current)
- int getHeightHelper (const BSTreeNode *current) const
- int getCountHelper (const BSTreeNode *current) const

Protected Attributes

- BSTreeNode * **root**

4.1.1 Constructor & Destructor Documentation

4.1.1.1 template<typename DataType , class KeyType > BSTree< DataType, KeyType >::BSTree ( )

This function is the default constructor for the BSTree class

This function will set the root of the BSTreeNode Class of the BSTree Class to NULL.

Parameters

none	

Returns

This function does not return anything.

Precondition

The BSTree is empty and was just created or needs to be modified.

Postcondition

The BSTree will not have the BSTreeNode root set to NULL

4.1.1.2 template<typename DataType , class KeyType > BSTree< DataType, KeyType >::BSTree (const BSTree< DataType, KeyType > & *other*)

This function is the copy constructor for the BSTree class.

This function will first set the root to NULL and then call the recursive function insert, with the parameters as root of the class and source.root.

Parameters

const	BSTree <DataType, KeyType> &other, which takes a BSTree by reference so that it can be used to copy the expression tree from the source to this object.

Returns

This function does not return anything.

Precondition

none

Postcondition

This function will create a copy of the parameter of BSTree& other and make the copy to this object.

4.1.1.3 template<typename DataType , class KeyType > BSTree< DataType, KeyType >::~BSTree ( )

This function is the destructor for the BSTree class.

This function will call the clear function to dynamically deallocate the memory for the current object.

Parameters

none	

Returns

This function does not return anything.

Precondition

An Exprtree object.

Postcondition

This BinaryTree will be cleared through dynamically memory allocation.

4.1.2 Member Function Documentation

4.1.2.1 template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::clear ( )

This function is the clear for the BSTree class.

This function will call the clearHelper function to dynamically deallocate the memory for the current object.

Parameters

none	

Returns

This function does not return anything.

Precondition

An Exprtree object.

Postcondition

The BinaryTree will be cleared through dynamically memory allocation.

4.1.2.2 template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::clearHelper (BSTreeNode *& *current*) [protected]

This function is the clearHelper for the BSTree class.

If current is not NULL, the function calls itself with current -> left as the parameter and then calls itself against with current -> right as the parameter. The function then deletes the current node and sets it to NULL.

Parameters

BSTreeNode	*& current, the current BSTreeNode to be passed so that it could be deleted from the BSTree.

Returns

This function does not return anything.

Precondition

An Exprtree object.

Postcondition

The BinaryTree will be cleared through dynamically memory allocation.

4.1.2.3 template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::copyHelper (BSTreeNode *& *dest*, BSTreeNode * *src*) [protected]

This function is the copyHelper function for the BSTree class, which is the helper function for the constructors for me.

This function will take in a pointer by reference and a pointer and take the source and modify the current to be the same as the soure. This is done by checking if the source is NULL, if it is, then return from the function. If Source is not NULL, then, we set the current with the ExprTreeNode param constructor with source's dataItem and its left and right set to NULL. I then recursively call the function till there is nothing left to insert, with dest -> left, src -> left and dest -> right and src -> right passed as the parameters.

Parameters

BSTreeNode	*&& dest, which takes a BSTreeNode pointer by reference so that it can be modified
BSTreeNode	*src, which takes a BSTreeNode pointer so that it can be used to set the other pointer passed to be reference as.

Returns

This function does not return anything.

Precondition

The BSTreeNode*& dest is empty and was just created or needs to be copied from the BSTreeNode* Src

Postcondition

The BSTreeNode*& dest will have a copy of the function of BSTreeNode*source.

4.1.2.4 template<typename DataType , class KeyType > int BSTree< DataType, KeyType >::getCount ( ) const

This function is the getHeight for the BSTree class.

The function checks to see if the tree is empty, if it is, the function returns 0, else the function returns getCountHelper with root passed as the parameter.

Parameters

none	

Returns

The function returns the number of data items in the tree.

Precondition

A BSTree object.

Postcondition

The function returns the number of data items in the tree.

4.1.2.5 template<typename DataType , class KeyType > int BSTree< DataType, KeyType >::getCountHelper (const BSTreeNode * *current*) const [protected]

This function is the getCountHelper for the BSTree class.

The function creates local int count that is set to 1, to account for the root node. The function checks to see if current -> left is not NULL, and it if is not the function sets count += recursive function call with current -> left passed as the parameter. The function checks to see if current -> right is not NULL, and it if is not the function sets count += recursive function call with current -> right passed as the parameter. The function always returns count.

Parameters

const	BSTreeNode * current, the current node to look at / observer to find the number of data items in the tree.

Returns

The function returns the number of data items in the tree.

Precondition

A BSTree object.

Postcondition

The function returns the number of data items in the tree.

4.1.2.6 template<typename DataType , class KeyType > int BSTree< DataType, KeyType >::getHeight ( ) const

This function is the getHeight for the BSTree class.

The function checks to see if the tree is empty, if it is, the function returns 0, else the function returns getHeightHelper with root passed as the parameter.

Parameters

none	

Returns

The function returns the height of the tree.

Precondition

A BSTree object.

Postcondition

The function returns the height of the tree.

4.1.2.7 template<typename DataType , class KeyType > int BSTree< DataType, KeyType >::getHeightHelper (const BSTreeNode * *current*) const [protected]

This function is the getHeightHelper for the BSTree class.

The function creates local int Lheight and Rheight variables and sets them to 0. If the current Node -> left is not NULL, put Lheight += call function with current -> left as the passed parameter. If the current Node -> right is not NULL, put Rheight += call function with current -> right as the passed parameter. If Lheigh is larger than Rheight the function returns Lheight + 1 for the size, which includes the first node. If Rheight is larger than or equal to Lheight the function returns Rheight + 1 for the size, which includes the first node. Otherwise the function will return one for the root node.

Parameters

const	BSTreeNode * current, the current node to look at / observer to find the height of the tree.

Returns

The function returns the height of the tree.

Precondition

A BSTree object.

Postcondition

The function returns the height of the tree.

4.1.2.8 template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::insert (const DataType & *newDataItem*)

This function is the insert function for the BSTree class, which inserts a new dataItem into the binary search true.

This function will call the insertHelper function with root and newDataItem passed as the parameters for the helper function.

Parameters

const	DataType& newDataItem, which is the dataItem passed by reference to be inserted.

Returns

This function does not return anything.

Precondition

A BSTree object to be modified.

Postcondition

The new DataItem will be inserted into the tree, if there is a dataItem with that already it replaces it anyways.

4.1.2.9 template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::insertHelper (BSTreeNode *& *current,* const DataType & *newDataItem*) [protected]

This function is the insertHelper function for the BSTree class, which is the helper function for the insert function.

If the current BSTreeNode current is NULL, the function creates a new node with the dataItem as the parameter and NULL, NULL as the other two, and return. If the DataItem key is smaller than the current dataItem key, the function calls itself with the parameters as current -> left and the dataitem. If the dataItem key is larger than the current dataItem key, the function calls itself with the parameters as current -> right and the dataitem. If the dataItem key is equal to the current dataItem key, the function inserts the newDataItem into current dataItem.

Parameters

BSTreeNode	*& current, which takes a ExpTreeNode pointer by reference so that it can be modified
const	DataType & newDataItem , dataItem passed by reference to be inserted.

Returns

This function does not return anything.

Precondition

A BSTree object to be modified.

Postcondition

A modified BSTree with the dataItem inserted into the BSTree.

4.1.2.10 template<typename DataType , class KeyType > bool BSTree< DataType, KeyType >::isEmpty ( ) const

This function is the isEmpty for the BSTree class.

The function checks to see if the root of the tree is NULL, and if it is NULL, the function returns true. Else the function returns false.

Parameters

none	

Returns

The function returns true or false based on if the tree is empty or not.

Precondition

An Exprtree object.

Postcondition

The function returns true or false based on if the tree is empty or not.

4.1.2.11 template<typename DataType , class KeyType > BSTree< DataType, KeyType > & BSTree< DataType, KeyType >::operator= (const BSTree< DataType, KeyType > & *other*)

This function is the overloaded assignment operator for the BSTree class.

This function will check if the current object is not the other parameter. The function will first clear the object, and then call the recursive function insert, with the parameters as root of the class and source.root.

Parameters

const	BSTree <DataType, KeyType> &other, which takes a BSTree by reference so that it can be used to copy the expression tree from the source to this object.

Returns

This function does not return anything.

Precondition

An object that is not the same as the object that was passed as the parameter for the copy constructor to be modified.

Postcondition

This function will create a copy of the parameter of BSTree& other and make the copy to this object.

4.1.2.12 template<typename DataType , class KeyType > bool BSTree< DataType, KeyType >::remove (const KeyType & *deleteKey*)

This function is the remove function for the BSTree class.

This function will call the removeHelper function with root and the passed parameter deleteKey as the parameters.

Parameters

const	KeyType &delete key, which is the key to be deleted from the BSTree.

Returns

This function returns true if the deletekey was found and deleted.

Precondition

An Exprtree object.

Postcondition

This function will delete the key if it is found and return true.

4.1.2.13 template<typename DataType , class KeyType > bool BSTree< DataType, KeyType >::removeHelper (BSTreeNode *& *current,* const KeyType & *deleteKey*) [protected]

This function is the removeHelper function for the BSTree class.

This function will return false if the current BSTreeNode is NULL. If the current dataItem key is smaller than the deleteKey, the function will return itself with current -> right and deleteKey as the passed parameters. If the current dataItem key is larger than the deleteKey, the function will return itself with current -> left and deleteKey as the passed parameters. If the current dataItem key is equal to the delete key, the function will check for the cases to delete the node. If the node has no children, the function will delete current, set current to NULL and return true. If the node has a left child, the function will set a temp node to equal current, set current to current -> left, delete temp and return true. If the node has a right child, the function will set a temp node to equal current, set current to current -> right, delete temp and return true. If the node has two children, the function sets the temp node to current -> left. Then in a while loop, the function will traverse the tree for current -> right until it finds the last one. The function then sets current -> dataItem to temp -> dataItem and calls itself with current -> left and delete key as the parameters and returns true when it succeeds with the recursion.

Parameters

const	KeyType &delete key, which is the key to be deleted from the BSTree.
BSTreeNode	*& current, which is the current tree node to be observed or deleted if it is the key.

Returns

This function returns true if the deletekey was found and deleted.

Precondition

An Exprtree object.

Postcondition

This function will delete the key if it is found and return true.

4.1.2.14 template<typename DataType , class KeyType > bool BSTree< DataType, KeyType >::retrieve (const KeyType & *searchKey,* DataType & *searchDataItem*) const

This function is the retrieve function for the BSTree class, which copies the dataItem to searchDataItem and returns true.

This function will call the insertHelper function with root and searchDataItem passed as the parameters for the helper function.

Parameters

const	KeyType& searchKey, which is the KeyType to be found in the BSTree.
const	DataType& searchDataItem, which holds the copied data of the searchKey if it is found.

Returns

This function does returns true or false, based on if the function dataItem is found.

Precondition

A BSTree object.

Postcondition

searchDataItem contains the data of the searchKey if the searchKey was found.

4.1.2.15 template<typename DataType , class KeyType > bool BSTree< DataType, KeyType >::retrieveHelper (BSTreeNode * *current,* const KeyType & *searchKey,* DataType & *searchDataItem*) const [protected]

This function is the retrieveHelper function for the BSTree class, which copies the dataItem to searchDataItem and returns true.

If the current BSTreeNode is null, the function returns false. If the searchKey is smaller than the current dataItem key, the function calls itself with current -> left, searchKey and searchDataItem as the parameters. If the searchKey is larger than the current dataItem key, the function calls itself with current -> right, searchKey and searchDataItem as the parameters. If the searchKey is equal to the current dataItem key, the function sets searchDataItem to the current dataItem and returns true.

Parameters

BSTreeNode	* current, which is the current BSTree Node that will be observed to search for the key.
const	KeyType& searchKey, which is the KeyType to be found in the BSTree.
const	DataType& searchDataItem, which holds the copied data of the searchKey if it is found.

Returns

This function does returns true or false, based on if the function dataItem is found.

Precondition

A BSTree object.

Postcondition

searchDataItem contains the data of the searchKey if the searchKey was found.

4.1.2.16 template<typename DataType , typename KeyType > void BSTree< DataType, KeyType >::showHelper (BSTreeNode * *p,* int *level*) const [protected]

This function is the showHelper function for the BSTree class

The function will iterate through the loop, and print out the leaves and branches of the Binary search tree.

Parameters

BSTreeNode	*p, which is the node to start the printing at.
int	level, the level to start the printing of the trees and branches at.

Returns

This function does not return anything.

Precondition

none

Postcondition

The contents of the BSTree will be printed to the terminal in the way that it should be.

4.1.2.17 template<typename DataType , typename KeyType > void BSTree< DataType, KeyType >::showStructure ( ) const

This function is the showStructure function for the BSTree class

The function will make sure that the tree is not empty. If it is, the function will print empty tree. Other wise it will call the showHelper function with root, and 1 passed as the parameters.

Parameters

none	

Returns

This function does not return anything.

Precondition

none

Postcondition

The contents of the BSTree will be printed to the terminal in the way that it should be.

4.1.2.18 template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::writeKeys ( ) const

This function is the writeKeys function for the BSTree class.

This function will call the writeKeysHelper function with root and the passed parameter if the BSTree is not empty.

Parameters

none	

Returns

This function returns nothing..

Precondition

An Exprtree object that is not empty.

Postcondition

This function will output the keys of the dataItem in the binary search tree.

4.1.2.19 template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::writeKeysHelper (const BSTreeNode * *current*) const [protected]

This function is the writeKeysHelper function for the BSTree class.

The function checks to see if the current pointer is not null. If it isnt, the function calls itself with current -> left and prints out itself's item. Then the function calls itself with current -> right.

Parameters

const	BSTreeNode * current, which is the current tree node to print out the dataItem.

Returns

This function returns nothing..

Precondition

An Exprtree object that is not empty.

Postcondition

This function will output the keys of the dataItem in the binary search tree.

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

- BSTree.h
- BSTree.cpp

4.2 BSTree< DataType, KeyType >::BSTreeNode Class Reference

Public Member Functions

- BSTreeNode (const DataType &nodeDataItem, BSTreeNode *leftPtr, BSTreeNode *rightPtr)

Public Attributes

- DataType **dataItem**
- BSTreeNode * **left**
- BSTreeNode * **right**

4.2.1 Constructor & Destructor Documentation

4.2.1.1 template<typename DataType , class KeyType > BSTree< DataType, KeyType >::BSTreeNode::BSTreeNode (const DataType & *nodeDataItem,* BSTreeNode * *leftPtr,* BSTreeNode * *rightPtr*)

This function is the BSTreeNode param constructoror the BSTree class

This function will initialize the BSTreeNode to hold a certain dataItem, and point to the left and right of its branch/leaf, depending on if they exist.

Parameters

const	DataType &nodeDataItem, which will be the dataItem that the node holds.
BSTreeNode	*leftPtr, which will be the node's left branch/leaf
BSTreeNode	*rightPtr, which will be the node's right branch/leaf

Returns

This function does not return anything.

Precondition

none

Postcondition

modifies the TreeNode to hold a dataitem and point to the left and right based on the parameters passed.

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

- BSTree.h
- BSTree.cpp

4.3 HashTable< DataType, KeyType > Class Template Reference

Public Member Functions

- HashTable (int initTableSize)
- HashTable (const HashTable &other)
- HashTable & operator= (const HashTable &other)
- ~HashTable ()
- void insert (const DataType &newDataItem)
- bool remove (const KeyType &deleteKey)
- bool retrieve (const KeyType &searchKey, DataType &returnItem) const
- void clear ()
- bool isEmpty () const
- void showStructure () const
- double **standardDeviation** () const

Private Member Functions

- void copyTable (const HashTable &source)

Private Attributes

- int **tableSize**
- BSTree< DataType, KeyType > * **dataTable**

4.3.1 Constructor & Destructor Documentation

4.3.1.1 template<typename DataType , typename KeyType > HashTable< DataType, KeyType >::HashTable (int *initTableSize*)

This function is the param constructor for the HashTable class.

This function will set the tableSize of the HashTable equal to the integer parameter passed. This function will also create a dataTable (array) of Binary Search Trees with the size of tableSize.

Parameters

int	initTableSize, which is the size of the HashTable

Returns

This function does not return anything.

Precondition

The Hashtable is empty and was just created or needs to be modified.

Postcondition

The HashTable will be intialized to be the size of the integer parameter passed.

4.3.1.2 template<typename DataType , typename KeyType > HashTable< DataType, KeyType >::HashTable (const HashTable< DataType, KeyType > & *other*)

This function is the copy constructor for the HashTable class.

This function will first call the copyTable function with other passed in as the parameter to create a copy of the other HashTable.

Parameters

const	HashTable<DataType, KeyType>& other, which is the other HashTable that is to be copied so that this object is the copy of the other./

Returns

This function does not return anything.

Precondition

none

Postcondition

This function will create a copy of the parameter of HashTable& other and make the copy to this object.

4.3.1.3 template<typename DataType , typename KeyType > HashTable< DataType, KeyType >::~HashTable ( )

This function is the destructor for the HashTable class.

This function will call the clear function to clear the data for the current object.

Parameters

none	

Returns

This function does not return anything.

Precondition

A HashTable to be cleared.

Postcondition

This HashTable will be cleared.

4.3.2 Member Function Documentation

4.3.2.1 template<typename DataType , typename KeyType > void HashTable< DataType, KeyType >::clear ( )

This function is the clear for the HashTable class.

This function will loop from index 0 to the size of the tableSize. In this for loop, the function iterates through each index of the HashTable and clears the object in the index. It does this by calling the clear function for each Binary Search Tree ADT within the index of the HashTable.

Parameters

none	

Returns

This function does not return anything.

Precondition

An HashTable object.

Postcondition

HashTable will be cleared through dynamically memory allocation.

4.3.2.2 template<typename DataType , typename KeyType > void HashTable< DataType, KeyType >::copyTable (const HashTable< DataType, KeyType > & *source*) [private]

This function is the copyTable function of the HashTable class. The function sets tableSize to the source's tableSize. The function then allocates memory to create an array of Binary Search Tree's equal to the tableSize. The function goes from index 0 to the tableSize and copies the data of the source array index to the current index.

Parameters

const	HashTable<DataType, KeyType>& source, which is the other HashTable to copy

Returns

None

Precondition

None

Postcondition

This HashTable will have the same data as the source HashTable.

4.3.2.3 template<typename DataType , typename KeyType > void HashTable< DataType, KeyType >::insert (const DataType & *newDataItem*)

This function is the insert function for the HashTable class, which inserts a new dataItem into the HashTable.

This function creates a local variable called index, and sets it to 0. Index is then set to the newDataItem's function called hash with the dataItem's getKey with modulus size of tableSize since we have to recircle back to the index of the table based on the bounds of the size of the HashTable. The function then inserts the newDataItem to the index of HashTable that we calculated.

Parameters

const	DataType& newDataItem, which is the dataItem passed by reference to be inserted.

Returns

This function does not return anything.

Precondition

A HashTable object to be modified.

Postcondition

The new DataItem will be inserted into the HashTable based on the index we calculate.

4.3.2.4 template<typename DataType , typename KeyType > bool HashTable< DataType, KeyType >::isEmpty ( ) const

This function is the isEmpty function for the HashTable class.

This function will loop from index 0 to the size of the tableSize. In the loop the function checks each index/Binary Search Tree and checks if it is Empty or not. This does this by calling the isEmpty function for the BST. The function returns false it isn't, otherwise the function returns true if all of it is cleared.

Parameters

none	

Returns

This function returns true if the HashTable is empty, otherwise it returns false.

Precondition

An HashTable object.

Postcondition

Returns true if the HashTable is empty, otherwise it returns false.

4.3.2.5 template<typename DataType , typename KeyType > HashTable< DataType, KeyType > & HashTable< DataType, KeyType >::operator= (const HashTable< DataType, KeyType > & *other*)

This function is the overloaded assignment operator for the HashTable class.

This function will check if the current object is not the other parameter. Then the function will first clear the object. This function will then call the copyTable function with other passed in as the parameter to create a copy of the other HashTable.

Parameters

const	HashTable<DataType, KeyType>& other, which is the other HashTable that is to be copied so that this object is the copy of the other./

Returns

This function returns a pointer to this object.

Precondition

none

Postcondition

This function will create a copy of the parameter of HashTable& other and make the copy to this object.

4.3.2.6 template<typename DataType , typename KeyType > bool HashTable< DataType, KeyType >::remove (const KeyType & *deleteKey*)

This function is the remove function for the HashTable class, which removes a deleteKey from the HashTable.

This function loops from index 0 to the size of the tableSize. In this for loop, the function checks to see if the deleteKey is removed from the HashTable. the function does this by looping until the HashTable is able to do this by calling the remove function from the Binary Search Tree ADT with the deleteKey passed as the parameter. If that statement is true, the function returns true. If that statement is not true and the for loop ends. The function returns false since the Key was never found or deleted.

Parameters

const	KeyType& deleteKey, which is the Key passed by reference to be deleted.

Returns

This function returns true if the KeyType was removed successfully from the HashTable.

Precondition

A HashTable object to be modified.

Postcondition

The deleteKey of the same value in the HashTable will be deleted.

4.3.2.7 template<typename DataType , typename KeyType > bool HashTable< DataType, KeyType >::retrieve (const KeyType & *searchKey,* DataType & *returnItem*) const

This function is the retrieve function for the HashTable class, which copies the dataItem to searchDataItem and returns true.

This function will loop from index 0 to the size of the tableSize. In this for loop, the function checks to see if the SearchKey is found in the HashTable. The function does this by checking the binary tree of each index to see if the searchKey is found in there. It does that by calling the retrieve function from the Binary Search Tree ADT of each index with the parameters of searchKey and returnItem passed as the parameters. The function checks to see if that process will return true. If it does, the function itself returns true. If the for loop ends and that process is never true, the function will return false.

Parameters

const	KeyType& searchKey, which is the KeyType to be found in the HashTable.
const	DataType& searchDataItem, which holds the copied data of the searchKey if it is found.

Returns

This function does returns true or false, based on if the function dataItem is found.

Precondition

A HashTable object.

Postcondition

searchDataItem contains the data of the searchKey if the searchKey was found.

4.3.2.8 template<typename DataType , typename KeyType > void HashTable< DataType, KeyType >::showStructure ( ) const

This function is the showStructure function for the HashTable class

The function will iterate through the loop, and print out the indexes of the HashTable.

Parameters

none	

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

- HashTable.h
- HashTable.cpp
- show10.cpp

4.4 TestData Class Reference

Public Member Functions

- TestData ()
- void setKey (const string &newKey)
- string getKey () const
- void setValue (const string &newValue)
- string getValue () const
- void **setKey** (const string &newKey)
- string **getKey** () const
- int **getValue** () const

Static Public Member Functions

- static unsigned int **hash** (const string &str)
- static unsigned int **hash** (const string &str)

Private Attributes

- string **key**
- string **value**
- int **value**

Static Private Attributes

- static int **count** = 0

4.4.1 Constructor & Destructor Documentation

4.4.1.1 TestData::TestData ( )

This function is the constructor for the TestData class.

This function does nothing.

Parameters

none	

Returns

This function does not return anything.

Precondition

nothing.

Postcondition

nothing.

4.4.2 Member Function Documentation

4.4.2.1 string TestData::getKey ( ) const

This function is the getKey for the TestData class.

This function returns the key(user) string.

Parameters

none	

Returns

This function returns the key(user) string.

Precondition

nothing.

Postcondition

This function returns the key(user) string.

4.4.2.2 int TestData::getValue ( ) const

This function is the getValue function for the TestData class.

This function returns the Value(password) string.

Parameters

none	

Returns

This function returns the Value(password) string.

Precondition

nothing.

Postcondition

This function returns the Value(password) string.

4.4.2.3 void TestData::setKey (const string & *newKey*)

This function is the setKey function for the TestData class.

This function will set the string key equal to newKey.

Parameters

const	string& newKey, which is the string to be user of the class.

Returns

This function does not return anything.

Precondition

A TestData class.

Postcondition

Key will now be equal to newKey.

4.4.2.4 void TestData::setValue (const string & *newValue*)

This function is the setValue function for the TestData class.

This function will set the string value equal to newValue.

Parameters

const	string& newValue, which is the string to be password of the class.

Returns

This function does not return anything.

Precondition

A TestData class.

Postcondition

Value will now be equal to newValue.

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

- login.cpp
- test10.cpp

5 File Documentation

5.1 BSTree.cpp File Reference

This program will implement an Binary Search Tree ADT using a linked tree structure.

```
#include "BSTree.h"
```

5.1.1 Detailed Description

This program will implement an Binary Search Tree ADT using a linked tree structure.

Author

Kripash Shrestha

Version

1.0

The specifications of the program are instructed and documented on Lab 9 Binary Search Tree ADT of C++ Data Structures: A Laboratory Course Third Edition by Brandle, Geisler, Roberge and Whittington

Date

Thursday, October 26, 2017

5.2 HashTable.cpp File Reference

This program will implement a HashTable using a Binary Search Tree ADT.

```
#include "HashTable.h"
```

5.2.1 Detailed Description

This program will implement a HashTable using a Binary Search Tree ADT.

Author

Kripash Shrestha

Version

1.0

The specifications of the program are instructed and documented on Lab 10 C++ Data Structures: A Laboratory Course Third Edition by Brandle, Geisler, Roberge and Whittington

Date

Wednesday, November 8, 2017

5.3 login.cpp File Reference

This program will implement a login simulator using HashTables.

```
#include <iostream>
#include <string>
#include <fstream>
#include "HashTable.cpp"
```

Classes

- class TestData

Functions

- int **main** ()

5.3.1 Detailed Description

This program will implement a login simulator using HashTables.

Author

Kripash Shrestha

Version

1.0

The specifications of the program are instructed and documented on Lab 10 C++ Data Structures: A Laboratory Course Third Edition by Brandle, Geisler, Roberge and Whittington

Date

Wednesday, November 8, 2017

Index