TITLE

AUTHOR
Version
CREATEDATE

Table of Contents

Table of contents

Class Index

Class List

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

File Index

File List

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

Class Documentation

Account Struct Reference

Public Member Functions

- int **getKey** () const

Static Public Member Functions

- static unsigned int **hash** (const int &key)

Public Attributes

- int **acctNum**
- float **balance**

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

- example1.cpp

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

Protected Member Functions

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

Protected Attributes

- **BSTreeNode** * **root**

Constructor & Destructor Documentation

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

Default constructor for **BSTree**

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

Creates an empty binary search tree

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

Copy constructor for **BSTree**

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

Initializes the binary search tree to be equivalent to the other **BSTree** object parameter

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

Destructor for **BSTree**

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

Deallocates (frees) the memory used to store the binary search tree.

Member Function Documentation

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

Removes all the data items in the binary search tree.

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

This binary search tree will have nothing in it and all memory will be deallocated.

template<typename DataType , typename KeyType > void BSTree< DataType, KeyType >::clearHelper (BSTreeNode *& *p*) `[protected]`

Recursively assist the clear method by deleting nodes one by one. Go as far as left as possible, go as far right as possible, delete the child nodes, then go back to the previous stack call and delete the child nodes which now have no other children

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

This binary search tree will have nothing in it and all memory will be deallocated.

template<typename DataType , typename KeyType > void BSTree< DataType, KeyType >::copyHelper (BSTreeNode *& *current*, BSTreeNode * *other*) `[protected]`

Helper function for copy constructor and assignment operator. Copies the contents, recursively, from one **BSTree** to this one

Parameters:

current	Current node of this **BSTree**
other	Current node of other **BSTree**

Returns:

None

Precondition:

None

Postcondition:

Copies other **BSTree** to current **BSTree**

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

Return the count of the number of data items in the binary search tree.

Parameters:

None	

Returns:

Number of data items in the binary search tree

Precondition:

None

Postcondition:

The contents of this **BSTree** will be unchanged.

template<typename DataType , typename KeyType > int BSTree< DataType, KeyType >::getCountHelper (const BSTreeNode * *p*) const `[protected]`

Recursive helper function for getCount Check every node on the left side, check every node on the right side. Add one for each that isn't null, return total. Return the count of the number of data items in the binary search tree.

Parameters:

None	

Returns:

Number of data items in the binary search tree

Precondition:

None

Postcondition:

The contents of this **BSTree** will be unchanged.

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

Calculate and return the height of the **BSTree**

Parameters:

None	

Returns:

The height of the binary search tree

Precondition:

None

Postcondition:

The contents of this **BSTree** will be unchanged.

template<typename DataType , typename KeyType > int BSTree< DataType, KeyType >::getHeightHelper (const BSTreeNode * *p*) const `[protected]`

Recursive helper function for getHeight If left not NULL, go down left side; if right not NULL, go down right side. Once at the bottom, return 1, then as we go back up compare whether the left or right side has a higher value to determine what to return Calculate and return the height of the **BSTree**

Parameters:

None	

Returns:

The height of the binary search tree

Precondition:

None

Postcondition:

The contents of this **BSTree** will be unchanged.

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

Inserts newDataItem into the binary search tree. If a data item with the same key as newDataItem already exists in the tree, then updates that data item with newDataItem.

Parameters:

newDataItem	item to be inserted into binary search tree

Returns:

None

Precondition:

None

Postcondition:

Another item will be added to this binary search tree if the passed in data is new.

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

Recursive helper for insert method. If null, create new node Else continue down tree until we find a null node If data greater than current val, go down right If data less than current val, go down left

Inserts newDataItem into the binary search tree. If a data item with the same key as newDataItem already exists in the tree, then updates that data item with newDataItem.

Parameters:

p	current node being evaluated for whether or not to insert into
newDataItem	item to be inserted into binary search tree

Returns:

None

Precondition:

None

Postcondition:

Another item will be added to this binary search tree if the passed in data is new.

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

Return if tree is empty or not.

Parameters:

None	

Returns:

True if the binary search tree is empty. Otherwise, returns false.

Precondition:

None

Postcondition:

The contents of this binary search tree will not be changed.

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

Overloaded assignment operator for **BSTree**

Parameters:

other	**BSTree** object to be set equal to

Returns:

BSTree A reference to this **BSTree** object

Precondition:

None

Postcondition:

Sets the binary search tree to be equivalent to the other **BSTree** object parameter and returns a reference to this object

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

Deletes the data item with the key deleteKey from the binary search tree. If this data item is found, then deletes it from the tree and returns true. Otherwise return false.

Parameters:

deleteKey	key to be deleted

Returns:

True if item is found and deleted; false otherwise.

Precondition:

None

Postcondition:

The deleteKey will be deleted from the tree if it exists in the tree. Otherwise the tree will be unchanged.

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

Recursive helper function for remove If no children, delete the node If one child, set child to current position then delete node If two children, find the predecessor to replace node, then delete node Deletes the data item with the key deleteKey from the binary search tree. If this data item is found, then deletes it from the tree and returns true. Otherwise return false.

Parameters:

p	current node being looked for deletion
deleteKey	key to be deleted

Returns:

True if item is found and deleted; false otherwise.

Precondition:

None

Postcondition:

The deleteKey will be deleted from the tree if it exists in the tree. Otherwise the tree will be unchanged.

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

Searches the binary search tree for the data item with key searchKey. If this data item is found, then copies the data item to searchDataItem and return true. Otherwise returns false with searchDataItem

Parameters:

searchKey	key to be searched for
searchDataItem	data to be updated if key found

Returns:

True if data item found; false otherwise

Precondition:

None

Postcondition:

The contents of this tree will not be changed

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

Recursive helper function for retrieve. If current value greater than search key, go down left If current value less than search key, go down right If null, return false Searches the binary search tree for the data item with key searchKey. If this data item is found, then copies the data item to searchDataItem and return true. Otherwise returns false with searchDataItem

Parameters:

p	current node being checked for if it is the searchKey
searchKey	key to be searched for
searchDataItem	data to be updated if key found

Returns:

True if data item found; false otherwise

Precondition:

None

Postcondition:

The contents of this tree will not be changed

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

Recursive helper for showStructure. Outputs the subtree whose root node is pointed to by p.

Parameters:

p	**BSTreeNode** currently being outputted
level	the level of this node within the tree

Returns:

None

Precondition:

None

Postcondition:

The contents of thi **BSTree** will be unchanged.

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

Outputs the keys in a binary search tree. The tree is output rotated counterclockwie 90 degrees from its conventional orientation using a "reverse" inorder traversal. This operation is intended for testing and debugging purposes only.

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

The contents of this **BSTree** will be unchanged.

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

Outputs the keys of the data items in the binary search tree. The keys are output in ascending order on one line, separated by spaces.

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

The contents of this **BSTree** will be unchanged.

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

Recursive helper function for writeKeys Outputs the keys of the data items in the binary search tree. The keys are output in ascending order on one line, separated by spaces.

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

The contents of this **BSTree** will be unchanged.

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

- BSTree.h
- **BSTree.cpp**

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**

Constructor & Destructor Documentation

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

Default constructor for **BSTreeNode**

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

Creates an empty binary search tree

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

- BSTree.h
- **BSTree.cpp**

Data Struct Reference

Public Member Functions

- void **setKey** (string newKey)
- string **getKey** () const

Static Public Member Functions

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

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

- test10std.cpp

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

Constructor & Destructor Documentation

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

HashTable constructor. Sets table size, and initializes dataTable as an array of size tableSize of BSTrees

Parameters:

initTableSize	: Table size

Returns:

none

Precondition:

none

Postcondition:

tableSize will be set, and dataTable will be initialized

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

HashTable copy constructor Utilizes copyTable method to do its dirty work.

Parameters:

other	: **HashTable** of which we are creating a copy of

Returns:

None

Precondition:

None

Postcondition:

This **HashTable** will have the same contents as our other parameter

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

Deallocate all the memory in this **HashTable**. Use the clear method to do our dirty work.

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

This **HashTable** will be free of memory

Member Function Documentation

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

Clears all the data/memory of this **HashTable** / deallocates all the memory of this **HashTable**. Goes through each element in the array and uses **BSTree**'s clear method on each element.

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

The memory of this **HashTable** will be deallocated.

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

Insert a new data item into the **HashTable**. Determine the index of this item based on its hash modulus'd by table size. Then use **BSTree**'s insert method on the index selected

Parameters:

newDataItem	: New data item to be inserted into the **HashTable**

Returns:

None

Precondition:

None

Postcondition:

The newDataItem will be a part of this **HashTable**

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

Returns whether or not this Hashtable is empty by checking whether the dataTable is NULL or not.

Parameters:

None	

Returns:

bool : True if dataTable is NULL, false if dataTable is not NULL

Precondition:

None

Postcondition:

The contents of this **HashTable** will not be changed.

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

HashTable Operator = Overload If the HashTables are already equal, don't do anything. Otherwise, clear out this **HashTable** and use copyTable to do our dirty work.

Parameters:

other	: **HashTable** of which we are settings ourselves equal to

Returns:

HashTable& : This **HashTable**

Precondition:

None

Postcondition:

This **HashTable** will be equal to our other parameter

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

Remove a delete key from the **HashTable** if it exists within the table. Use **BSTree**'s remove method on each element of our table. If the **BSTree** remove method returns true (meaning it deleted a key), return true. Otherwise, if we get through all of the elements in the table, return false.

Parameters:

deleteKey	: key that we are looking to delete from this table

Returns:

bool : true if we successfully deleted the key, false if the key does not exist in this **HashTable**

Precondition:

None

Postcondition:

If the deleteKey passed in exists in the list, it will be removed from the list. Otherwise, everything in the list will not be modified.

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

Retrieve a data item from this **HashTable** based on a search key. For each element in the table, use **BSTree**'s retrieve method. If this ever returns true, this method will return true. Otherwise, if the loop goes through all elements in the table, return false - indicating that the searchKey does not exist in this **HashTable**.

Parameters:

searchKey	: Key being searched through our **HashTable** for
returnItem	: **Data** Item that will either be unmodified if the searchKey cannot be found, or will be changed to the dataItem that corresponds to the searchKey if it is found.

Returns:

bool : True if the searchKey exists in this **HashTable**, false if the searchKey does not exist in this **HashTable**

Precondition:

None

Postcondition:

The contents of this **HashTable** will not be modified.

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

Shows the structure of this **HashTable**. Goes through each element of the **HashTable** and uses **BSTree**'s writeKeys method.

Parameters:

None	

Returns:

None

Precondition:

None

Postcondition:

The contents of this **HashTable** will be unchanged.

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

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

TestData Class Reference

Public Member Functions

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

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

- **login.cpp**
- test10.cpp

File Documentation

BSTree.cpp File Reference

This program will implement a Binary Search Tree.
`#include "BSTree.h"`

Detailed Description

This program will implement a Binary Search Tree.

Author:
Tim Kwist

Version:
1.0

The specifications of this program are defined by C++ **Data** Structures: A Laboratory Course (3rd edition) by Brandle, JGeisler, Roberge, Whittington, lab 9.

Date:
Wednesday, October 8, 2014

HashTable.cpp File Reference

This program will implement a Hash Table.
`#include "HashTable.h"`

Detailed Description

This program will implement a Hash Table.

Author:
Tim Kwist

Version:
1.0

The specifications of this program are defined by C++ **Data** Structures: A Laboratory Course (3rd edition) by Brandle, JGeisler, Roberge, Whittington, lab 10.

Date:
Wednesday, October 29, 2014

login.cpp File Reference

This program will implement the Exercise 1 of Lab 10 Hash Table.

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

Classes

- class **TestData**

Functions

- int **main** ()

Detailed Description

This program will implement the Exercise 1 of Lab 10 Hash Table.

Author:

Tim Kwist

Version:

1.0

The specifications of this program are defined by C++ **Data** Structures: A Laboratory Course (3rd edition) by Brandle, JGeisler, Roberge, Whittington, lab 10.

Date:

Wednesday, October 29, 2014

Function Documentation

int main ()

This method will simulate a login authenticator. The list of login/password combos will be read from a file, password.dat, and stored in a hash table. Then the user will be allowed to input usernames and passwords which will be ran against the hash table. If valid login/password, 'Authentication successful' will be output. Otherwise, 'Authentication failure' will be output.

Parameters:

None	

Returns:

None

Precondition:

password.dat is a valid file to open

Postcondition:

None

Index

INDEX