Project 10 (Hash Table ADT)

AUTHOR

Version 1.00

10/28/2014

Table of Contents

Project 10 (Binary Search Tree ADT)

This program will implement a Hash Table using an array of binary search trees.

Author:

Saharath Kleips

The specifications of this project match those of the book C++ Data Structures - A Laboratory Course (3rd Edition) Project 10. A Hash Table maps a unique key onto a specific location in an array. The generation of these keys is called a hash function. Certain hash functions will occasionally generate identical indexes for different keys, to prevent this, Binary Search Trees are used to implement chaining.

Todo List

Member BSTree< DataType, KeyType >::writeLessThan (const KeyType &searchKey) const

Function implementation.

Member BSTree< DataType, KeyType >::writeLessThanHelper (const KeyType &searchKey, BSTreeNode *p) const

Function implementation.

Member HashTable< DataType, KeyType >::standardDeviation () const

Implement function.

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**

Detailed Description

Definition at line 8 of file example1.cpp.

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

1 example1.cpp

BSTree< DataType, KeyType > Class Template Reference

Classes

class BSTreeNode

Public Member Functions

BSTree (const BSTree< DataType, KeyType > &other)

> *The copy constructor that initializes this BSTree to be equivalent to the other BSTree object parameter.*

BSTree & operator= (const BSTree< DataType, KeyType > &other)

> *The overloaded assignment operator that sets this BSTree to be equivalent to the other BSTree object parameter and returrns a reference to this object.*

~BSTree ()

> *The destructor that deallocates the memory used to store this BSTree.*

void insert (const DataType &newDataItem)

> *Inserts newDataItem into this BSTree.*

bool retrieve (const KeyType &searchKey, DataType &searchDataItem) const

> *Searches this BSTree for the data item with key searchKey.*

bool remove (const KeyType &deleteKey)

> *Deletes the data item with key deleteKey from this BSTree.*

void writeKeys () const

> *Outputs the keys of the data items in this BSTree.*

void clear ()

> *Removes all data items in this BSTree.*

bool isEmpty () const

> *Returns true if this BSTree is empty.*

void **showStructure** () const
int getHeight () const

> *Returns the height of this BSTree.*

int getCount () const

> *Returns the count of the number of data items in this BSTree.*

void writeLessThan (const KeyType &searchKey) const

> *Outputs all keys in this BSTree that are less than searchKey.*

Protected Member Functions

void copyHelper (BSTreeNode *&p, BSTreeNode *other)

> *Recursive helper function.*

void insertHelper (const DataType &newDataItem, BSTreeNode *&p)

> *Recursive helper function.*

bool retrieveHelper (const KeyType &searchKey, DataType &searchDataItem, BSTreeNode *p) const

> *Recursive helper function.*

bool removeHelper (const KeyType &deleteKey, BSTreeNode *&p)

> *Recursive helper function.*

void writeKeysHelper (BSTreeNode *p) const

> *Recursive helper function.*

void clearHelper (BSTreeNode *&p)

Recursive helper function.

int getHeightHelper (BSTreeNode *p, int currentLevel) const

Recursive helper function.

int getCountHelper (BSTreeNode *p) const

Recursive helper function.

void writeLessThanHelper (const KeyType &searchKey, BSTreeNode *p) const

Recursive helper function.

void **showHelper** (BSTreeNode *p, int level) const

Protected Attributes

BSTreeNode * **root**

Detailed Description

template<typename DataType, class KeyType>class BSTree< DataType, KeyType >

Definition at line 20 of file BSTree.h.

Constructor & Destructor Documentation

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

The copy constructor that initializes this BSTree to be equivalent to the other BSTree object parameter.

Precondition:

other is a valid BSTree.

Postcondition:

This BSTree will be a deep copy of the other BSTree.

Parameters:

other	is the BSTree that this BSTree will be made equivalent to.

BSTree<DataType,KeyType>::operator=(const BSTree<DataType,KeyType>&)

Definition at line 46 of file BSTree.cpp.

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

The destructor that deallocates the memory used to store this BSTree.

Postcondition:

This BSTree will be an empty, deallocated, BSTree.

See also:

BSTree<DataType,KeyType>::clear()

Definition at line 95 of file BSTree.cpp.

Member Function Documentation

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

Removes all data items in this BSTree.

Postcondition:

This BSTree will be an empty, deallocated, BSTree.

See also:

BSTree<DataType,KeyType>::clearHelper(BSTreeNode*& p)

Definition at line 304 of file BSTree.cpp.

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

Recursive helper function.

Parameters:

See also:

p	is the current node to check against.

BSTree<DataType,KeyType>::clear()

Definition at line 314 of file BSTree.cpp.

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

Recursive helper function.

Parameters:

See also:

p	is the node to copy into (destination node).
other	is the node to copy from (source node).

BSTree<DataType,KeyType>::operator=(const BSTree<DataType,KeyType>&)

Definition at line 77 of file BSTree.cpp.

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

Returns the count of the number of data items in this BSTree.

Returns:

An integer representation of how many data items are in this BSTree.

See also:

BSTree<DataType,KeyType>::getCountHelper(BSTreeNode*)

Definition at line 386 of file BSTree.cpp.

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

Recursive helper function.

Parameters:

See also:

p	is the current node to check against.

BSTree<DataType,KeyType>::getCount()

Definition at line 396 of file BSTree.cpp.

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

Returns the height of this BSTree.

Returns:

An integer representation of the height of this BSTree. see BSTree<DataType,KeyType>::getHeightHelper(BSTreeNode*, int)

Definition at line 350 of file BSTree.cpp.

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

Recursive helper function.

Parameters:

See also:

p	is the current node to check against.
currentLevel	is the level of the current node.

BSTree<DataType,KeyType>::getHeight()

Definition at line 361 of file BSTree.cpp.

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

Inserts newDataItem into this BSTree.

If a data item with the same key as newDataItem already exists in this tree, then updates that data item with newDataItem.

Postcondition:

newDataItem will be inserted with respect to left and right BSTreeNodes.

Parameters:

See also:

newDataItem	is the data item to be inserted into this BSTree.

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

Recursive helper function.

Parameters:

Definition at line 107 of file BSTree.cpp.

newDataItem	is the data item to insert into the BSTree.
p	is the current node to check against.

BSTree<DataType,KeyType>::insert(const DataType&)

Definition at line 118 of file BSTree.cpp.

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

Returns true if this BSTree is empty.

Otherwise, returns false.

Returns:

True if this BSTree is empty. False if this BSTree is not empty.

Definition at line 338 of file BSTree.cpp.

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

The overloaded assignment operator that sets this BSTree to be equivalent to the other BSTree object parameter and returrns a reference to this object.

Precondition:

other is a valid BSTree.

Postcondition:

This BSTree will be a deep copy of the other BSTree

Parameters:

See also:

other	is this BSTree that this BSTree will be made equivalent to.

The reference to this BSTree.

See also:

BSTree<DataType,KeyType>::copyHelper(BSTreeNode*&, BSTreeNode*)

Definition at line 61 of file BSTree.cpp.

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

Deletes the data item with key deleteKey from this BSTree.

If the data item is found, then deletes it from the tree and returns true. Otherwise, returns false.

Postcondition:

This BSTree will no longer contain the data item with key deleteKey if found.

Parameters:

Returns:

deleteKey	is the key to search which data item to delete in this BSTree.

True if the data item is found. False if the data item is not found.

See also:

BSTree<DataType,KeyType>::removeHelper(const KeyType&, BSTreeNode*&)

Definition at line 200 of file BSTree.cpp.

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

Recursive helper function.

Parameters:

Returns:

deleteKey	is the key to compare all the nodes within BSTree to.
p	is the current node to check against.

BSTree<DataType,KeyType>::remove(const KeyType&)

Definition at line 211 of file BSTree.cpp.

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

Searches this BSTree for the data item with key searchKey.

If this data item is found, then copies the data item to searchDataItem and returns true. Otherwise, returns false with searchDataItem equal to null.

Postcondition:

searchDataItem will be the copied data item if the data item is found.

Parameters:

See also:

searchKey	is the key to search this BSTree for.
searchDataItem	is the data item that will contain the search key's data item if found.

True if the data item is found. False if the data item is not found.

See also:

BSTree<DataType,KeyType>::retrieveHelper(const KeyType&, DataType&, BSTreeNode*)

Definition at line 152 of file BSTree.cpp.

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

Recursive helper function.

Parameters:

Returns:

searchKey	is the key to compare all nodes within the BSTree to.
searchDataItem	will contain the search key's data item if found.
p	is the current node to check against.

BSTree<DataType,KeyType>::retrieve(const KeyType&, DataType&)

Definition at line 165 of file BSTree.cpp.

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

Outputs the keys of the data items in this BSTree.

The keys are output in ascending order on one line, separated by spaces.

Postcondition:

The keys of each data item are outputted to the console.

See also:

BSTree<DataType,KeyType>::writeKeysHelper(BSTreeNode*)

Definition at line 276 of file BSTree.cpp.

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

Recursive helper function.

Parameters:

See also:

p	is the current node to check against.

BSTree<DataType,KeyType>::writeKeys()

Definition at line 287 of file BSTree.cpp.

template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::writeLessThan (const KeyType & *searchKey*) const

Outputs all keys in this BSTree that are less than searchKey.

The keys are output in ascending order on one line, separated by spaces. searchKey does not need to be a key in this BSTree.

Postcondition:

The keys less than searchKey are outputted to the console.

Parameters:

See also:

searchKey	is the key to compare if this BSTree's keys are less than to.

BSTree<DataType,KeyType>::writeLessThanHelper(const keyType&, BSTreeNode*)

Todo:

Function implementation.

Definition at line 416 of file BSTree.cpp.

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

Recursive helper function.

Parameters:

See also:

searchKey	is the key to compare if this BSTree's keys are less than to.
p	is the current node to check against.

BSTree<DataType,KeyType>::writeLessThan(const KeyType&)

Todo:

Function implementation.

Definition at line 428 of file BSTree.cpp.

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

2 BSTree.h
3 BSTree.cpp
4 show9.cpp

BSTree< DataType, KeyType >::BSTreeNode Class Reference

Public Member Functions

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

> *The parameterized constructor that sets the BSTreeNode's data item to the value nodeDataItem, BSTreeNode's previous pointer to the value leftPtr, and BSTreeNode's next pointer to the value rightPtr.*

Public Attributes

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

Detailed Description

template<typename DataType, class KeyType>class BSTree< DataType, KeyType >::BSTreeNode

Definition at line 55 of file BSTree.h.

Constructor & Destructor Documentation

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

> The parameterized constructor that sets the BSTreeNode's data item to the value nodeDataItem, BSTreeNode's previous pointer to the value leftPtr, and BSTreeNode's next pointer to the value rightPtr.

Postcondition:

> This BSTreeNode will be a valid initialized BSTreeNode.

Parameters:

See also:

nodeDataItem	is the data to be stored within the node.
leftPtr	is the pointer to the previous BSTreeNode that this BSTreeNode is linked to.
rightPtr	is the pointer to the next BSTreeNode that this BSTreeNode is linked to.

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

5 BSTree.h
6 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)

Private Attributes

string **key**

Detailed Description

Definition at line 17 of file test10std.cpp.

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

7 test10std.cpp

HashTable< DataType, KeyType > Class Template Reference

Public Member Functions

HashTable (int initTableSize)

The parameterized constructor that creates an empty HashTable of size initTableSize.

HashTable (const HashTable &other)

The copy constructor that initializes this HashTable to be equivalent to the other HashTable.

HashTable & operator= (const HashTable &other)

The overloaded assignment operator that sets this HashTable to be equivalent to the other HashTable object parameter.

~HashTable ()

The destructor that deallocates the memory used to store this HashTable.

void insert (const DataType &newDataItem)

Inserts newDataItem into the appropriate binary search tree.

bool remove (const KeyType &deleteKey)

Removes the data item from this HashTable by searching for the data item with the key deleteKey.

bool retrieve (const KeyType &searchKey, DataType &returnItem) const

Searches for the data item from this HashTable with key searchkey.

void clear ()

Removes all data items from the HashTable.

bool isEmpty () const

Returns true if this HashTable is empty.

void **showStructure** () const
double standardDeviation () const

Computes the standard deviation for key distribution in the hash table and returns the result.

Private Member Functions

void copyTable (const HashTable &source)

Recursive helper function.

Private Attributes

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

Detailed Description

template<typename DataType, typename KeyType>class HashTable< DataType, KeyType >

Definition at line 15 of file HashTable.h.

Constructor & Destructor Documentation

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

The parameterized constructor that creates an empty HashTable of size initTableSize.

Postcondition:

This HashTable will be a valid empty HashTable of size initTableSize.

Parameters:

Definition at line 450 of file BSTree.cpp.

initTableSize	is the size of the table

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

The copy constructor that initializes this HashTable to be equivalent to the other HashTable.

Precondition:

other is a valid HashTable.

Postcondition:

This HashTable will be a deep copy of the other HashTable.

Parameters:

Definition at line 32 of file HashTable.cpp.

other	is the HashTable that this HashTable will be made equivalent to.

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

The destructor that deallocates the memory used to store this HashTable.

Postcondition:

This HashTable will be an empty, deallocated, HashTable.

See also:

HashTable<DataType,KeyType>::clear()

Definition at line 78 of file HashTable.cpp.

Member Function Documentation

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

Removes all data items from the HashTable.

Postcondition:

This HashTable will be an empty HashTable.

Definition at line 142 of file HashTable.cpp.

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

Recursive helper function.

Parameters:

Definition at line 45 of file HashTable.cpp.

other	is the HashTable to copy from (source HashTable).

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

Definition at line 179 of file HashTable.cpp.

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

Inserts newDataItem into the appropriate binary search tree.

If a data item with the same key as newDataItem already exists in the binary search tree, then update item with newDataItem. Otherwise, it inserts it in the binary search tree.

Postcondition:

newDataItem will be inserted into the HashTable.

Parameters:

See also:

newDataItem	is the data item to be inserted into this HashTable.

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

Returns true if this HashTable is empty.

Otherwise, returns false.

Returns:

True if this HashTable is empty. False if this HashTable is not empty.

Definition at line 154 of file HashTable.cpp.

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

The overloaded assignment operator that sets this HashTable to be equivalent to the other HashTable object parameter.

Precondition:

other is a valid HashTable.

Postcondition:

This HashTable will be a deep copy of the other HashTable.

Parameters:

Definition at line 91 of file HashTable.cpp.

other	is the HashTable table that this HashTable will be made equivalent to.

The reference to this HashTable.

See also:

HashTable<DataType,KeyType>::copyTable(const HashTable& source)

Definition at line 61 of file HashTable.cpp.

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

Removes the data item from this HashTable by searching for the data item with the key deleteKey.

If the data item is found, return true. Otherwise, return false.

Postcondition:

This HashTable will no longer contain the data item with key deleteKey.

Parameters:

Returns:

deleteKey	is the key to search which data item to delete in this HashTable.

True if the data item is found. False if the data item is not found.

Definition at line 108 of file HashTable.cpp.

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

Searches for the data item from this HashTable with key searchkey.

If the data item is found, then copy the data item into returnItem and returns true. Otherwise, returns false with returnItem undefined.

Postcondition:

searrchDataItem will be the copied data item if the data item is found.

Parameters:

Returns:

searchKey	is the key to search this HashTable for.

returnItem	is the data item that will contain the search key's data item if found.

True if the data item is found. False if the data item is not found.

Definition at line 128 of file HashTable.cpp.

template<typename DataType , typename KeyType > double HashTable< DataType, KeyType >::standardDeviation () const

Computes the standard deviation for key distribution in the hash table and returns the result.

Returns:

The standard deviation (Currently always -1.0).

Todo:

Implement function.

Definition at line 169 of file HashTable.cpp.

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

8 HashTable.h

9 HashTable.cpp

10 show10.cpp

Login Class Reference

Public Member Functions

void setKey (const string &username)

Mutator for username data of this Login.

void setPassword (const string &password)

Mutator for password data of this Login.

string getKey () const

Accessor for key data of this Login.

string getPassword () const

Accessor for password data of this Login.

Static Public Member Functions

static unsigned int hash (const string &s)

Generates a hash for a string value.

Private Attributes

string **key**
string **password**

Detailed Description

Definition at line 25 of file login.cpp.

Member Function Documentation

string Login::getKey () const

Accessor for key data of this Login.

Returns:

The key of this Login.

Definition at line 121 of file login.cpp.

string Login::getPassword () const

Accessor for password data of this Login.

Returns:

The password of this Login.

Definition at line 130 of file login.cpp.

unsigned int Login::hash (const string & *s*) `[static]`

Generates a hash for a string value.

Parameters:

Returns:

s	is the string to generate the hash value from.

The hash value as an unsigned integer.

Definition at line 140 of file login.cpp.

void Login::setKey (const string & *username*)

Mutator for username data of this Login.

Postcondition:

username is updated with the new username.

Parameters:

Returns:

username	is the new username of this login.

void Login::setPassword (const string & *password*)

Mutator for password data of this Login.

Postcondition:

password is updated with the new password.

Parameters:

Definition at line 102 of file login.cpp.

password	is the new password of this login.

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

11 login.cpp

TestData Class Reference

Public Member Functions

void **setKey** (const string &newKey)
string **getKey** () const
int **getValue** () const

Static Public Member Functions

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

Private Attributes

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

Static Private Attributes

static int **count** = 0

Detailed Description

Definition at line 8 of file test10.cpp.

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

12 test10.cpp

File Documentation

BSTree.cpp File Reference

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

```
#include "BSTree.h"
#include "show9.cpp"
```

Detailed Description

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

Author:

Saharrath Kleips

The specifications of this project match those of the book C++ Data Structures - A Laboratory Course (3rd Edition) Project 9. Data items within the data structure form a binary tree. Data items are of generic type DataType and each data item has a key of generic type KeyType. For each data item D in the tree, all the data items in D's left subtree have keys that are less than D's key and all the data items in D's right subtree have keys that are greater than D's key. /n Note: Binary Search Tree == BSTree, Binary Search Tree Node == BSTreeNode

Definition in file BSTree.cpp.

HashTable.cpp File Reference

```
#include "HashTable.h"
#include "show10.cpp"
```

Detailed Description

Definition in file HashTable.cpp.

login.cpp File Reference

This program tests the username / password functionality of a Hash Table.

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

Classes

class Login

Functions

int main ()

The main entry point of this program.

Detailed Description

This program tests the username / password functionality of a Hash Table.

Author:

Saharath Kleips

The specifications of this project match those of the book C++ Data Structures - A Laboratory Course (3rd Edition) Project 10. This program will load username / password sets from a file and insert them into the Hash Table. There should be one username / password set (separated by 1 tab) per line. The program will ask for a login and password then will determine if authentication is successful or not.

Definition in file login.cpp.

Index

INDE