Lab 10 Hash Table - Ernest Landrito

Generated by Doxygen 1.8.5

Tue Nov 5 2013 22:18:41

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 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

3.1.1 Member Function Documentation

3.1.1.1 int Account::getKey ( ) const [inline]

3.1.1.2 static unsigned int Account::hash (const int & *key*) [inline], [static]

3.1.2 Member Data Documentation

3.1.2.1 int Account::acctNum

3.1.2.2 float Account::balance

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

- example1.cpp

3.2 BSTree< DataType, KeyType > Class Template Reference

`#include <BSTree.h>`

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 recInsert (const DataType &newDataItem, BSTreeNode *&node)
- bool recRetrieve (const KeyType &searchKey, DataType &searchDataItem, BSTreeNode *node) const
- bool recRemove (const KeyType &deleteKey, BSTreeNode *&node)
- void recWriteKeys (BSTreeNode *node) const
- void recClear (BSTreeNode *&node)
- void recGetHeight (BSTreeNode *node, int currentLevel, int &maxLevel) const
- int recGetCount (BSTreeNode *node) const
- void recCopy (BSTreeNode *&node, BSTreeNode *otherNode)
- void recWriteLessThan (const KeyType &searchKey, BSTreeNode *node) const
- void showHelper (BSTreeNode *p, int level) const

Protected Attributes

- BSTreeNode * root

3.2.1 Constructor & Destructor Documentation

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

Precondition

New BSTree Class

Postcondition

root pointing to null

Algorithm:

- point root to null

Exceptional/Error Conditions:

- none

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

Precondition

new BSTree class

Postcondition

a deep copy of source BSTree

Parameters

source	Source BSTree to be deep copied

Algorithm:

- Use overloaded assignment operator to copy the data from the source to this class

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

Precondition

a BSTreeClass

Postcondition

deallocated BSTreeClass

Algorithm:

- Use clear member function to deallocate the tree

3.2.2 Member Function Documentation

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

Precondition

a BSTreeClass

Postcondition

a deallocated tree

Algorithm:

- Use recClear member function to make the expression tree

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

Precondition

a BSTreeClass

Postcondition

returns the number of nodes in the tree

Returns

returns the height of the tree

Algorithm:

- use recGetCount member function

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

Precondition

a BSTreeClass

Postcondition

returns the height of the tree

Returns

returns the height of the tree

Algorithm:

- use recGetHeight member function

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

Precondition

a BSTreeClass

Postcondition

a new node inserted into the BSTree

Parameters

newDataItem	Data to be inserted into the BSTree

Algorithm:

- Use recInsert member function

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

Precondition

a BSTreeClass

Postcondition

returns if the tree is empty

Returns

Returns true if root is pointing to NULL

Algorithm:

- return true if root is pointing to NULL

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

Precondition

a BSTreeClass

Postcondition

a deep copy of source BSTree

Parameters

source	Source BSTree to be deep copied

Algorithm:

- Use recCopy to make a copy of each node in the BSTree

3.2.2.7 template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::recClear (BSTreeNode *& *node*) [protected]

Precondition

an BSTreeClass

Postcondition

deallocated BSTreeClass

Parameters

node	the node to be deleted.

Algorithm:

- Post order traverse the tree, deleting the node when bottom of tree is reached

3.2.2.8 template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::recCopy (BSTreeNode *& *node,* BSTreeNode * *otherNode*) [protected]

Precondition

an BSTreeClass

Postcondition

a deep copy of source BSTree

Parameters

node	location in this tree
otherNode	location in source tree

Algorithm:

- if the source node is not at a NULL location -create temp nodes ptrs and copy those from source -create a node and point to the temp nodes

3.2.2.9 template<typename DataType , class KeyType > int BSTree< DataType, KeyType >::recGetCount (BSTreeNode * *node*) const [protected]

Precondition

a BSTreeNode

Postcondition

returned amount of nodes in the tree

Returns

Amount of nodes in the tree

Parameters

node	the location in the tree

Algorithm:

- if at a null position return 0
- else return 1 plus recursive call on the children

3.2.2.10 template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::recGetHeight (BSTreeNode * *node,* int *currentLevel,* int & *maxLevel*) const [protected]

Precondition

a BSTreeNode

Postcondition

returned value of the height of the tree

Parameters

node	the location in the tree
currentLevel	the current level of the tree
maxLevel	reference to the value to be returned

Algorithm:

- if at a leaf check if level is greater than max, if so return the value
- if not at a leaf go to children

3.2.2.11 template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::recInsert (const DataType & *newDataItem,* BSTreeNode *& *node*) [protected]

Precondition

an new BSTreeNode

Postcondition

initialized BSTreeNode

Parameters

nodeDataItem	Element to be stored in the node
*leftPtr	Pointer to the next left node
*rightPtr	Pointer to the next right node

Algorithm:

- Assign parameters to the corresponding variables

Precondition

an new BSTreeNode

Postcondition

inserted Value in the tree

Parameters

newDataItem	new node data to be put into the tree
*node	Pointer to current node

Algorithm:

- check if youre at a leaf, if so insert the data
- check if you found a value of the same key, if so update
- check if your key is greater than current, if so traverse right
- else traverse left

3.2.2.12 template<typename DataType , class KeyType > bool BSTree< DataType, KeyType >::recRemove (const KeyType & *deleteKey,* BSTreeNode *& *node*) [protected]

Precondition

a BSTreeNode

Postcondition

node removed from the list

Returns

Returns if successful

Parameters

deleteKey	The key to be searched for in the tree.
node	the location in the tree to be checked

Algorithm:

- check if youre at the end of the list
- check if you're at the right location if so move on
- check if you are at a leaf if so delete the node
- else check if there is one child that is null
- if there is one child that is null delete the node
- else find the node just less than the node, replace it and delete the node
- if reaches end of the list return false

3.2.2.13 template<typename DataType , class KeyType > bool BSTree< DataType, KeyType >::recRetrieve (const KeyType & *searchKey,* DataType & *searchDataItem,* BSTreeNode * *node*) const [protected]

Precondition

a BSTreeNode

Postcondition

a copy of the search data item

Returns

Returns if successful

Parameters

searchKey	The key to be searched for in the tree.
searchDataItem	The reference variable to store the found data
node	the location in the tree to be checked

Algorithm:

- check if youre at the end of the list
- check if you are at the right key if so assign data
- else check if key is less than location, if so call function to left
- else call function to the right
- if reaches end of the list return false

3.2.2.14 template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::recWriteKeys (BSTreeNode * *node*) const [protected]

Precondition

a BSTreeNode

Postcondition

printed list of the keys

Parameters

node	the location in the tree to be checked

Algorithm:

- in order traversal of the tree printing the nodes

3.2.2.15 template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::recWriteLessThan (const KeyType & *searchKey,* BSTreeNode * *node*) const [protected]

Precondition

a BSTreeNode

Postcondition

printed list of the keys

Parameters

node	the location in the tree to be checked

Algorithm:

- in order traversal of the tree printing the nodes only printing the trees if the key is less than the search key

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

Precondition

a BSTreeClass

Postcondition

a data item removed from the list if found

Returns

Returns true if deleted, false if not

Parameters

deleteKey	The key value to be searched for in the tree

Algorithm:

- Use recRemove member function

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

Precondition

a BSTreeClass

Postcondition

a copy of the dataItem found

Returns

Returns true if found, false if not

Parameters

searchKey	The key value to be searched for in the tree
searchDataItem	The reference value to be changed if found

Algorithm:

- Use recRetrieve member function

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

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

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

Precondition

a BSTreeClass

Postcondition

The key values of the list printed in ascending order

Algorithm:

- Use recWriteKeys member function

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

Precondition

a BSTreeClass

Postcondition

prints the key values less than the one given

Parameters

searchKey	the key value to upper bound the list

Algorithm:

- use recWriteLessThan member function

3.2.3 Member Data Documentation

3.2.3.1 template<typename DataType, class KeyType> BSTreeNode* BSTree< DataType, KeyType >::root [protected]

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

- BSTree.h
- BSTree.cpp

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

`#include <BSTree.h>`

Public Member Functions

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

Public Attributes

- DataType dataItem
- BSTreeNode * left
- BSTreeNode * right

3.3.1 Constructor & Destructor Documentation

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

3.3.2 Member Data Documentation

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

3.3.2.2 template<typename DataType, class KeyType> BSTreeNode* BSTree< DataType, KeyType >::BSTreeNode::left

3.3.2.3 template<typename DataType, class KeyType> BSTreeNode * BSTree< DataType, KeyType >::BSTreeNode::right

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

- BSTree.h

3.4 Data Struct Reference

Public Member Functions

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

Static Public Member Functions

- static unsigned int hash (const string &str)

3.4.1 Member Function Documentation

3.4.1.1 string Data::getKey ( ) const [inline]

3.4.1.2 static unsigned int Data::hash (const string & *str*) [inline], [static]

3.4.1.3 void Data::setKey (string *newKey*) [inline]

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

- test10std.cpp

3.5 HashTable< DataType, KeyType > Class Template Reference

`#include <HashTable.h>`

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

3.5.1 Constructor & Destructor Documentation

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

Precondition

New HashTable Class

Postcondition

dynamically allocated array of size initTableSize

Parameters

initTableSize	the size of the array to be formed

Algorithm:

- assign table size to the init tablesize
- dynamically declare array

Exceptional/Error Conditions:

- none

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

Precondition

new HashTable class

Postcondition

a deep copy of source HashTable

Parameters

other	Source HashTable to be deep copied

Algorithm:

- Use overloaded assignment operator to copy the data from the source to this class

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

Precondition

a HashTable

Postcondition

deallocated HashTable

Algorithm:

- Use clear member function to deallocate the HashTable

3.5.2 Member Function Documentation

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

Precondition

a HashTable

Postcondition

a deallocated HashTable

Algorithm:

- For every location in the table, clear the BST

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

Precondition

a HashTable

Postcondition

a new node inserted into the BSTree at the appropriate array location of the HashTable

Parameters

newDataItem	Data to be inserted into the HashTable

Algorithm:

- Hash the data item's key and use the return value as a location
- Insert into BST the new data at that location

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

Precondition

a HashTable

Postcondition

returns if the HashTable is empty

Returns

Returns true if every point in the array contains an empty tree

Algorithm:

- return false if any location of the data table is not empty
- otherwise, return true

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

Precondition

a BSTreeClass

Postcondition

a deep copy of source HashTable

Parameters

source	Source BSTree to be deep copied

Algorithm:

- Use copyTable to make a copy of the HashTable

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

Precondition

a BSTreeClass

Postcondition

a BST removed from the HashTable if

Returns

Returns true if deleted, false if not

Parameters

deleteKey	The key value to be searched for in the HashTable

Algorithm:

- Use BST remove to remove the key at the Hashed value of the delete Key

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

Precondition

a HashTable

Postcondition

a copy of the dataItem found

Returns

Returns true if found, false if not

Parameters

searchKey	The key value to be searched for in the tree
returnItem	The reference value to be changed if found

Algorithm:

- Hash the searchKey
- Use BST retrieve to get the data from the BST at location
- Return success of BST retrieve

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

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

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

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

3.6 LoginInfo Class Reference

Public Member Functions

- void setInfo (const string &newUsername, const string &newPassword)
- string getKey () const
- string getPassword () const

Static Public Member Functions

- static unsigned int hash (const string &str)

3.6.1 Member Function Documentation

3.6.1.1 string LoginInfo::getKey ( ) const

Precondition

a LoginInfo class

Postcondition

a returned key of the data

Returns

The key returned

Algorithm:

- return key

3.6.1.2 string LoginInfo::getPassword ( ) const

Precondition

a LoginInfo class

Postcondition

a returned password of the data

Returns

The password returned

Algorithm:

- return password

3.6.1.3 unsigned int LoginInfo::hash (const string & *str*) [static]

Precondition

a LoginInfo class

Postcondition

a returned sum of the value of the chars in the string

Returns

sum of the value of the chars in the string

Algorithm:

- for each char of the string, add the value to val
- return val

3.6.1.4 void LoginInfo::setInfo (const string & *newUsername,* const string & *newPassword*)

Precondition

LoginInfo class

Postcondition

Values set

Parameters

newUsername	string key is going to be assigned
newPassword	string password is going to be assigned

Algorithm:

- assign values

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

- login.cpp

3.7 TestData Class Reference

Public Member Functions

- TestData ()
- TestData (const TestData &source)
- void setKey (const string &newKey)
- string getKey () const
- int getValue () const

Static Public Member Functions

- static unsigned int hash (const string &str)

3.7.1 Constructor & Destructor Documentation

3.7.1.1 TestData::TestData ( )

3.7.1.2 TestData::TestData (const TestData & *source*)

3.7.2 Member Function Documentation

3.7.2.1 string TestData::getKey ( ) const

3.7.2.2 int TestData::getValue ( ) const

3.7.2.3 unsigned int TestData::hash (const string & *str*) [static]

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

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

- test10.cpp

Chapter 4

File Documentation

4.1 BSTree.cpp File Reference

```
#include <stdexcept>
#include <iostream>
#include "BSTree.h"
```

4.2 BSTree.h File Reference

```
#include <stdexcept>
#include <iostream>
```

Classes

- class BSTree< DataType, KeyType >
- class BSTree< DataType, KeyType >::BSTreeNode

4.3 example1.cpp File Reference

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

Classes

- struct Account

Functions

- int main ()

4.3.1 Function Documentation

4.3.1.1 int main ( )

4.4 HashTable.cpp File Reference

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

4.5 HashTable.h File Reference

```
#include <stdexcept>
#include <iostream>
#include "BSTree.cpp"
```

Classes

- class HashTable< DataType, KeyType >

4.6 login.cpp File Reference

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

Classes

- class LoginInfo

Functions

- int main ()

4.6.1 Function Documentation

4.6.1.1 int main ( )

4.7 show10.cpp File Reference

4.8 test10.cpp File Reference

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

Classes

- class TestData

Functions

- void print_help ()
- int main (int argc, char **argv)

4.8.1 Function Documentation

4.8.1.1 int main (int *argc,* char ** *argv*)

4.8.1.2 void print_help ( )

4.9 test10std.cpp File Reference

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

Classes

- struct Data

Functions

- int main ()

4.9.1 Function Documentation

4.9.1.1 int main ( )

Index