

Lab 10 Hash Table - Ernest Landrito

Generated by Doxygen 1.8.5

Tue Nov 5 2013 22:18:41

Contents

1	Class Index	1
1.1	Class List	1
2	File Index	3
2.1	File List	3
3	Class Documentation	5
3.1	Account Struct Reference	5
3.1.1	Member Function Documentation	5
3.1.1.1	getKey	5
3.1.1.2	hash	5
3.1.2	Member Data Documentation	5
3.1.2.1	acctNum	5
3.1.2.2	balance	5
3.2	BSTree< DataType, KeyType > Class Template Reference	5
3.2.1	Constructor & Destructor Documentation	6
3.2.1.1	BSTree	6
3.2.1.2	BSTree	7
3.2.1.3	~BSTree	7
3.2.2	Member Function Documentation	7
3.2.2.1	clear	7
3.2.2.2	getCount	7
3.2.2.3	getHeight	8
3.2.2.4	insert	8
3.2.2.5	isEmpty	8
3.2.2.6	operator=	9
3.2.2.7	recClear	9
3.2.2.8	recCopy	9
3.2.2.9	recGetCount	10
3.2.2.10	recGetHeight	10
3.2.2.11	recInsert	11
3.2.2.12	recRemove	11

3.2.2.13	recRetrieve	12
3.2.2.14	recWriteKeys	12
3.2.2.15	recWriteLessThan	13
3.2.2.16	remove	13
3.2.2.17	retrieve	13
3.2.2.18	showHelper	14
3.2.2.19	showStructure	14
3.2.2.20	writeKeys	14
3.2.2.21	writeLessThan	14
3.2.3	Member Data Documentation	15
3.2.3.1	root	15
3.3	BSTree< DataType, KeyType >::BSTreeNode Class Reference	15
3.3.1	Constructor & Destructor Documentation	15
3.3.1.1	BSTreeNode	15
3.3.2	Member Data Documentation	15
3.3.2.1	dataItem	15
3.3.2.2	left	15
3.3.2.3	right	15
3.4	Data Struct Reference	15
3.4.1	Member Function Documentation	16
3.4.1.1	getKey	16
3.4.1.2	hash	16
3.4.1.3	setKey	16
3.5	HashTable< DataType, KeyType > Class Template Reference	16
3.5.1	Constructor & Destructor Documentation	16
3.5.1.1	HashTable	16
3.5.1.2	HashTable	17
3.5.1.3	~HashTable	17
3.5.2	Member Function Documentation	17
3.5.2.1	clear	17
3.5.2.2	insert	17
3.5.2.3	isEmpty	18
3.5.2.4	operator=	18
3.5.2.5	remove	18
3.5.2.6	retrieve	19
3.5.2.7	showStructure	19
3.5.2.8	standardDeviation	19
3.6	LoginInfo Class Reference	19
3.6.1	Member Function Documentation	20
3.6.1.1	getKey	20

3.6.1.2	getPassword	20
3.6.1.3	hash	20
3.6.1.4	setInfo	21
3.7	TestData Class Reference	21
3.7.1	Constructor & Destructor Documentation	21
3.7.1.1	TestData	21
3.7.1.2	TestData	21
3.7.2	Member Function Documentation	21
3.7.2.1	getKey	21
3.7.2.2	getValue	21
3.7.2.3	hash	21
3.7.2.4	setKey	21
4	File Documentation	23
4.1	BSTree.cpp File Reference	23
4.2	BSTree.h File Reference	23
4.3	example1.cpp File Reference	23
4.3.1	Function Documentation	24
4.3.1.1	main	24
4.4	HashTable.cpp File Reference	24
4.5	HashTable.h File Reference	24
4.6	login.cpp File Reference	24
4.6.1	Function Documentation	24
4.6.1.1	main	24
4.7	show10.cpp File Reference	24
4.8	test10.cpp File Reference	24
4.8.1	Function Documentation	25
4.8.1.1	main	25
4.8.1.2	print_help	25
4.9	test10std.cpp File Reference	25
4.9.1	Function Documentation	25
4.9.1.1	main	25
Index		26

Chapter 1

Class Index

1.1 Class List

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

Account	5
BSTree< DataType, KeyType >	5
BSTree< DataType, KeyType >::BSTreeNode	15
Data	15
HashTable< DataType, KeyType >	16
LoginInfo	19
TestData	21

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

BSTree.cpp	23
BSTree.h	23
example1.cpp	23
HashTable.cpp	24
HashTable.h	24
login.cpp	24
show10.cpp	24
test10.cpp	24
test10std.cpp	25

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

<i>source</i>	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 & newDataType)`

Precondition

a BSTreeClass

Postcondition

a new node inserted into the [BSTree](#)

Parameters

<i>newDataType</i>	Data to be inserted into the BSTree
--------------------	---

Algorithm:

- Use reInsert 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

<i>source</i>	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

<i>node</i>	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

<i>node</i>	location in this tree
<i>otherNode</i>	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

<i>node</i>	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

<i>node</i>	the location in the tree
<i>currentLevel</i>	the current level of the tree
<i>maxLevel</i>	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 >::reinsert (const DataType & newDataType, BSTreeNode *& node) [protected]`

Precondition

an new [BSTreeNode](#)

Postcondition

initialized [BSTreeNode](#)

Parameters

<i>nodeDataItem</i>	Element to be stored in the node
<i>*leftPtr</i>	Pointer to the next left node
<i>*rightPtr</i>	Pointer to the next right node

Algorithm:

- Assign parameters to the corresponding variables

Precondition

an new [BSTreeNode](#)

Postcondition

inserted Value in the tree

Parameters

<i>newDataItem</i>	new node data to be put into the tree
<i>*node</i>	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

<i>deleteKey</i>	The key to be searched for in the tree.
<i>node</i>	the location in the tree to be checked

Algorithm:

- check if you're 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

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

Algorithm:

- check if you're 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

<i>node</i>	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

<i>node</i>	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

<i>deleteKey</i>	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

<i>searchKey</i>	The key value to be searched for in the tree
<i>searchDataItem</i>	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

<i>searchKey</i>	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

<i>initTableSize</i>	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

<i>other</i>	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 & newDataType)`

Precondition

a [HashTable](#)

Postcondition

a new node inserted into the [BSTree](#) at the appropriate array location of the [HashTable](#)

Parameters

<i>newDataItem</i>	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

<i>source</i>	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

<i>deleteKey</i>	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

<i>searchKey</i>	The key value to be searched for in the tree
<i>returnItem</i>	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

<i>newUsername</i>	string key is going to be assigned
<i>newPassword</i>	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

- ~BSTree
 - BSTree, 7
- ~HashTable
 - HashTable, 17
- Account, 5
 - acctNum, 5
 - balance, 5
 - getKey, 5
 - hash, 5
- acctNum
 - Account, 5
- BSTree
 - ~BSTree, 7
 - BSTree, 6
 - BSTree, 6
 - clear, 7
 - getCount, 7
 - getHeight, 8
 - insert, 8
 - isEmpty, 8
 - operator=, 9
 - recClear, 9
 - recCopy, 9
 - recGetCount, 10
 - recGetHeight, 10
 - recInsert, 10
 - recRemove, 11
 - recRetrieve, 12
 - recWriteKeys, 12
 - recWriteLessThan, 13
 - remove, 13
 - retrieve, 13
 - root, 15
 - showHelper, 14
 - showStructure, 14
 - writeKeys, 14
 - writeLessThan, 14
- BSTree< DataType, KeyType >, 5
- BSTree< DataType, KeyType >::BSTreeNode, 15
- BSTree.cpp, 23
- BSTree.h, 23
- BSTree::BSTreeNode
 - BSTreeNode, 15
 - dataltem, 15
 - left, 15
 - right, 15
- BSTreeNode
 - BSTree::BSTreeNode, 15
- balance
 - Account, 5
- clear
 - BSTree, 7
 - HashTable, 17
- Data, 15
 - getKey, 16
 - hash, 16
 - setKey, 16
- dataltem
 - BSTree::BSTreeNode, 15
- example1.cpp, 23
 - main, 24
- getCount
 - BSTree, 7
- getHeight
 - BSTree, 8
- getKey
 - Account, 5
 - Data, 16
 - LoginInfo, 20
 - TestData, 21
- getPassword
 - LoginInfo, 20
- getValue
 - TestData, 21
- hash
 - Account, 5
 - Data, 16
 - LoginInfo, 20
 - TestData, 21
- HashTable
 - ~HashTable, 17
 - clear, 17
 - HashTable, 16
 - HashTable, 16
 - insert, 17
 - isEmpty, 18
 - operator=, 18
 - remove, 18
 - retrieve, 19
 - showStructure, 19
 - standardDeviation, 19
- HashTable< DataType, KeyType >, 16
- HashTable.cpp, 24
- HashTable.h, 24

- insert
 - BSTree, 8
 - HashTable, 17
- isEmpty
 - BSTree, 8
 - HashTable, 18
- left
 - BSTree::BSTreeNode, 15
- login.cpp, 24
 - main, 24
- LoginInfo, 19
 - getKey, 20
 - getPassword, 20
 - hash, 20
 - setInfo, 20
- main
 - example1.cpp, 24
 - login.cpp, 24
 - test10.cpp, 25
 - test10std.cpp, 25
- operator=
 - BSTree, 9
 - HashTable, 18
- print_help
 - test10.cpp, 25
- recClear
 - BSTree, 9
- recCopy
 - BSTree, 9
- recGetCount
 - BSTree, 10
- recGetHeight
 - BSTree, 10
- recInsert
 - BSTree, 10
- recRemove
 - BSTree, 11
- recRetrieve
 - BSTree, 12
- recWriteKeys
 - BSTree, 12
- recWriteLessThan
 - BSTree, 13
- remove
 - BSTree, 13
 - HashTable, 18
- retrieve
 - BSTree, 13
 - HashTable, 19
- right
 - BSTree::BSTreeNode, 15
- root
 - BSTree, 15
- setInfo
 - LoginInfo, 20
- setKey
 - Data, 16
 - TestData, 21
- show10.cpp, 24
- showHelper
 - BSTree, 14
- showStructure
 - BSTree, 14
 - HashTable, 19
- standardDeviation
 - HashTable, 19
- test10.cpp, 24
 - main, 25
 - print_help, 25
- test10std.cpp, 25
 - main, 25
- TestData, 21
 - getKey, 21
 - getValue, 21
 - hash, 21
 - setKey, 21
 - TestData, 21
 - TestData, 21
- writeKeys
 - BSTree, 14
- writeLessThan
 - BSTree, 14