PA08-L10

Generated by Doxygen 1.7.6.1

Mon Oct 27 2014 16:05:29

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 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

Protected Member Functions

- void showHelper (BSTreeNode *p, int level) const
- void insertHelper (BSTreeNode *&ptr, const DataType &newDataItem)
- bool removeHelper (BSTreeNode *&ptr, const KeyType &deleteKey)

- bool retrieveHelper (BSTreeNode *ptr, const KeyType &searchKey, DataType &searchDataItem) const
- void clearHelper (BSTreeNode *&ptr)
- void writeKeysHelper (BSTreeNode *ptr) const
- void copyHelper (BSTreeNode *&ptr, BSTreeNode *sourcePtr)
- int getHeightHelper (BSTreeNode *ptr) const
- int getCountHelper (BSTreeNode *ptr) const

Protected Attributes

- BSTreeNode * root

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

3.1.1 Constructor & Destructor Documentation

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

default constructor

Creates an empty binary search tree. set root to null

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

copy constructor

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

Parameters

other	reference to a BST to be copied from

set root to null

use copy helper to set values

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

destructor

Dellocates (frees) the memory used to store the binary search tree. clear values

3.1.2 Member Function Documentation

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

clear

Removes all data items in the binary search tree

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

clearHelper

Recursive helper for clear. Ends if at null pointer. Else, calls to remove children and deletes itself. Calls itself to remove all data items in the binary search tree

Parameters

ptr	BSTreeNode pointer to current node

if pointer is null

if data has children

clear left and right chilren

delete root

set to null

3.1.2.3 template<typename DataType , typename KeyType > void BSTree< DataType, KeyType >::copyHelper (BSTreeNode *& *ptr,* BSTreeNode * *sourcePtr*) [protected]

copyHelper

Sets this BS tree to be equivalent to the other BSTree parameter by calling itself to copy each node

Parameters

ptr	BSTreeNode pointer to current node to copy to
sourcePtr	BSTreeNode pointer to source's node to copy from

if node empty, end

copy value in source node

copy left and right values

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

getCount

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

Returns

int count of number of data items in BST

call helper to count data items

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

getCountHelper

Returns the count of number of data items in the BST

Parameters

ptr	BSTreeNode pointer to current node to copy to

Returns

int count of items in BST

base case - end of branch

recursive call - add 1 (this item) plus counts of left and right branches

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

getHeight

Returns the height of the binary search tree.

Returns

int height of BST

call helper to count height

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

getHeightHelper

Returns the height of the BST

Parameters

ptr	BSTreeNode pointer to current node to copy to

Returns

int height of BST

base case - end of branch

if left branch has greater height than right

return 1 (for this node) plus the height of left branch

otherwise

return 1 (for this node) plus the height of right branch

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

insert

Calls insertHelper to insert a new data item into BST. Inserts new data item into the BST. If a data item with the sane key as newDataItem already exists in the tree, then updates that data item with newDataItem.

Parameters

newDataItem	reference to the data to be inserted

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

insertHelper

Recursive helper for insert. Inserts new data item into the BST. If a data item with the sane key as newDataItem already exists in the tree, then updates that data item with newDataItem. Calls itself if data should go to right or left until a null is found.

Parameters

ptr	BSTreeNode pointer to current node
newDataItem	int value to be inserted

if current tree node is null

insert a new node with given data

if data to be inserted is less than current tree node

call insertHelper with node to the right

if data to be inserted is greater than current node

call insertHelper with node to the right

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

isEmpty

Returns true is the BST is empty. Otherwise, returns false.

Returns

bool if tree is empty or not

return true if root is null, false otherwise

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

assignment operator

Sets the BS tree to be equivalent to the other BSTree parameter and returns a reference to this object.

Parameters

other	reference to a BS tree to be copied from

Returns

BSTree& reference to this BS tree

if not same expression trees

clear values

copy values using copy helper

return this expression tree, dereferenced

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

remove

Calls removeHelper to delete the key passed. Deletes the data item with key deleteKey from the binary search tree. If the data item is found, then deletes it from the tree and returns true. Otherwise, returns false.

Parameters

deleteKey	a reference to the key to delete

Returns

bool true if data was found and removed, false otherwise

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

removeHelper

Recursive helper for remove. Calls itself to delete the key passed. Deletes the data item with key deleteKey from the binary search tree. If the data item is found, then deletes it from the tree and returns true. Otherwise, returns false.

Parameters

ptr	BSTreeNode pointer to current node
deleteKey	int value to be deleted

if ptr is null

value was not found

if value was found

case 1 - no children

delete node

set ptr to null

return that data was deleted

case 2 - 1 child

case 2l - left child

initialize temp node pointer

point temp to ptr

point ptr to its left child

delete temp (original ptr)

return that data was deleted

case 2r - right child

initialize temp node pointer

point temp to ptr

change ptr to its right child

delete temp (original ptr)

return that data was deleted

case 3 - 2 children

initialize a temp note pointer

set the temp pointer to ptr

point temp to its left child

until temp equals null

point temp to its right child

set ptr's data to that of temp's (change the value of the removed node to that of it's closest child)

call removeHelper to repeat on remaining children and return result

if the ptr's data is greater than the one to delete

call removeHelper to test child to left

if the ptr's data is less than the one to delete

call removeHelper to test child to right

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

retrieve

Calls retrieveHelper to find the data item passed. Searches the BST 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 and searchDataItem undefined.

Parameters

searchKey	a reference to the key searching for
searchDataItem	a reference to the data value to find

Returns

bool if value was found

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

retrieveHelper

Recursive helper for retrieve. Calls itself to find the data item passed. Searches the BST 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 and searchDataItem undefined.

Parameters

ptr	BSTreeNode pointer to current node
deleteKey	int value to be deleted

base cases

if current node is null

value was not found, return false

if search data item is found

set search data item, return true

recursive calls

if search item is less than pointer's

call self with node to the left

if search item is greater than pointer's

call self with node to the right

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

showHelper

Recursive helper for showStructure. Outputs the subtree whose root node is pointed to by p. Parameter level is the level of this node within the tree.

Parameters

p	pointer to current node
level	int count of number of levels currently

Loop counter

Output right subtree

Tab over to level

Output key

Output "connector"

Output left subtree

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

showStructure

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

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

writeKeys

Outputs the keys of the data items in the BST. The keys are output in ascending order on one line, seperated by spaces.

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

writeKeysHelper

Recursive helper for writeKeys. Outputs the keys of the data items in the BST. The keys are output in ascending order on one line, seperated by spaces.

Parameters

ptr	BSTreeNode pointer to current node

for each node that isn't empty

print nodes to left

print this node

print nodes to right

3.1.3 Member Data Documentation

3.1.3.1 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

3.2 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

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

3.2.1 Constructor & Destructor Documentation

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

constructor

Creates a binary search tree node

Parameters

nodeDataItem	reference to data to save to node
leftPtr	pointer to node to the left
rightPtr	pointer to node to the right

3.2.2 Member Data Documentation

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

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

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

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

- BSTree.h
- BSTree.cpp

3.3 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

Private Member Functions

- void copyTable (const HashTable &source)

Private Attributes

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

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

3.3.1 Constructor & Destructor Documentation

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

default constructor

Creates an empty hash table of size initTableSize. set the size of the hash table

create a table of BSTs

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

copy constructor

Initializes the hash table to be equivalent to the other hash table parameter.

Parameters

other	reference to a BST to be copied from

set table size

create table of BSTs

call copyTable to copy table

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

destructor

Dellocates (frees) the memory used to store the hash table. calls clear to delete data in table

3.3.2 Member Function Documentation

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

clear

Removes all data items in the hash table initialize variables

for each tree in table

clear values of that tree

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

copyTable

Copies data of one table to the other

Parameters

source	(const HashTable&) table to be copied from

initialize variables

for each tree in table

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

insert

Inserts newDataItem into the appropriate BST. If a data item with the same key as newDataItem already exists in the BST, then updates that data item with newDataItem. Otherwise, it inserts it into the BST.

Parameters

newDataItem	reference to the data to be inserted

get index of tree to insert to

insert into appropriate data table

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

isEmpty

Returns true if the hash table is empty. Otherwise, returns false.

Returns

bool if hash table is empty or not

initialize variables

for each tree in table

if tree is not empty return false

return true if all trees empty

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

assignment operator

Sets the hash table to be equivalent to the other hash table parameter and returns a reference to this object.

Parameters

other	reference to a hash table to be copied from

Returns

HashTable& reference to this hash table

if the tables are not the same instances

clear values in this table

set table size

create table of BSTs

call copyTable to copy other's data into this table

return reference to this table

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

remove

Searches the hash table for the data item with the key deleteKey. If the data item is found, then removes the data item and returns true. Otherwise, returns false.

Parameters

deleteKey	a reference to the key to delete

Returns

bool true if data was found and removed, false otherwise

initialize variables

for each tree in table

return true if deleteKey found

return false if deleteKey not found

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

retrieve

Searches the hash table for the data item with key searchKey. If the data item is found, then copies the data item to returnItem and returns true. Otherwise, returns false with returnItem undefined.

Parameters

searchKey	a reference to the key searching for (const KeyType&)
returnItem	a reference to the data value found (DataType&)

Returns

bool if value was found

initialize variables

for each tree in table

return true if retrieveKey found

return false if retrieveKey not found

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

showStructure

Outputs the trees in the hash table. Outputs "Empty hash table" if empty. This operation is intended for testing and debugging purposes only. for each tree in hash table

prints which tree in hash table

uses writeKeys of BST to print each tree

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

standardDeviation

Returns

double of standard deviation

3.3.3 Member Data Documentation

3.3.3.1 template<typename DataType , typename KeyType > BSTree<DataType, KeyType>* HashTable< DataType, KeyType >::dataTable [private]

3.3.3.2 template<typename DataType , typename KeyType > int HashTable< DataType, KeyType >::tableSize [private]

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

- HashTable.h
- HashTable.cpp

3.4 TestData Class Reference

data class used in hash table's binary search trees

Public Member Functions

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

  set key field
- void setPwd (const string &newPwd)

  return key field
- string getPwd () const

  set password field
- TestData ()
- void setKey (const string &newKey)
- string getKey () const
- int getValue () const

Static Public Member Functions

- static unsigned int hash (const string &str)

  return password field
- static unsigned int hash (const string &str)

Private Attributes

- string key
- string pwd

  (key) account username
- int value

Static Private Attributes

- static int count = 0

3.4.1 Detailed Description

data class used in hash table's binary search trees

3.4.2 Constructor & Destructor Documentation

3.4.2.1 TestData::TestData ( )

default constructor

Creates an instance of test data.

3.4.2.2 TestData::TestData ( )

3.4.3 Member Function Documentation

3.4.3.1 string TestData::getKey ( ) const

3.4.3.2 string TestData::getKey ( ) const

set key field

getKey

Returns the key of TestData

Returns

string of key

3.4.3.3 string TestData::getPwd ( ) const

set password field

getPwd

Returns the password of TestData

Returns

string of password

3.4.3.4 int TestData::getValue ( ) const

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

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

return password field

hash

Gets the index of where to insert passed in string using algorithm involving adding each character value

Parameters

string	of the key which decides the hash index

Returns

unsigned int containing hash index calculated

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

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

setKey

Sets the key of TestData to the key passed

Parameters

newKey	(const string&) to be assigned to key

3.4.3.9 void TestData::setPwd (const string & *newPwd*)

return key field

setPwd

Sets the password of TestData to the password passed

Parameters

newPwd	(const string&) to be assigned to password

3.4.4 Member Data Documentation

3.4.4.1 int TestData::count = 0 [static, private]

3.4.4.2 string TestData::key [private]

3.4.4.3 string TestData::pwd [private]

(key) account username

3.4.4.4 int TestData::value [private]

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

- login.cpp

- test10.cpp

Chapter 4

File Documentation

4.1 BSTree.cpp File Reference

```
#include <stdexcept>  #include <iostream>  #include "BS-
Tree.h"
```

4.1.1 Detailed Description

Author

CatherinePollock

Date

10/16/14

This is the implementation file for the BSTree.h file.

4.2 BSTree.h File Reference

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

Classes

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

4.3 HashTable.cpp File Reference

```
#include <stdexcept> #include <iostream> #include "Hash-
Table.h"
```

4.3.1 Detailed Description

Author

CatherinePollock

Date

10/27/14

This is the implementation file for the HashTable.h file.

4.4 HashTable.h File Reference

```
#include <stdexcept>  #include <iostream>  #include "BS-
Tree.cpp"
```

Classes

- class HashTable< DataType, KeyType >

4.5 login.cpp File Reference

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

Classes

- class TestData

  data class used in hash table's binary search trees

Functions

- int main ()

4.5.1 Detailed Description

Author

CatherinePollock

Date

10/27/14

This is the file used to read in account usernames and passwords from a file, prompt user for a username and password, and then authenticate the entered username and password, based on the accounts read in from file.

4.5.2 Function Documentation

4.5.2.1 int main ( )

main function

Reads in account information, prompts user for login information, and compares with saved data, printing results.

Returns

int success or failure

initialize variables

loop through each record in file

save account username

skip over spaces

save account password

insert data into hash table

clear data file flags

print the structure of names (keys) saved into hash table

prompt for username

prompt for password

check for username in table

if the name was found

check if password matches and print results

if the name was not found print failure

prompt for another login

continue until end of input

4.6 test10.cpp File Reference

```
#include <iostream>  #include <string>  #include "Hash-
Table.cpp"
```

Classes

- class TestData

  data class used in hash table's binary search trees

Functions

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

4.6.1 Function Documentation

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

4.6.1.2 void print_help ( )