PA07_LAB09_HENRY_HUFFMAN

Generated by Doxygen 1.7.6.1

Mon Oct 20 2014 11:39:42

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 AccountRecord Struct Reference

Public Attributes

- int acctID
- char firstName [nameLength]
- char lastName [nameLength]
- double balance

3.1.1 Member Data Documentation

3.1.1.1 int AccountRecord::acctID

3.1.1.2 double AccountRecord::balance

3.1.1.3 char AccountRecord::firstName

3.1.1.4 char AccountRecord::lastName

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

- database.cpp
- database.cs

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 showHelper (BSTreeNode *p, int level) const
- void assignmentHelper (BSTreeNode *&dest, BSTreeNode *src)
- bool removeHelper (BSTreeNode *&src, const KeyType &deleteKey)
- void insertHelper (BSTreeNode *&ptr, const DataType &newDataItem)
- bool retrieveHelper (BSTreeNode *ptr, const KeyType &searchKey, DataType &searchDataItem) const
- void writeHelper (BSTreeNode *ptr) const
- void clearHelper (BSTreeNode *&ptr)
- int countHelper (BSTreeNode *ptr) const
- int heightHelper (BSTreeNode *ptr) const
- void lessHelper (BSTreeNode *ptr, const KeyType &searchKey) const

Protected Attributes

- BSTreeNode * root

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

3.2.1 Constructor & Destructor Documentation

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

BSTree constructor

This constructor sets the root pointer of the initialized tree to null

Parameters

none	

Returns

none

Precondition

there will be an uninitialized tree

Postcondition

there will an initialized tree with the root pointer set to null

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

BSTree copy constructor

This constructor initializes a tree then copies the value of another tree. This process is completed by setting the root pointer to null, then calling the overloaded assigment operator to copy the values of a second tree.

Parameters

other	- a binary tree that is to be copied to the current tree

Returns

none

Precondition

there will be one unitialized tree, and a tree that has not been copied

Postcondition

there will be two trees with identical values

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

BSTree deconstructor

This function clears all the data inside the current tree. It first checks to see if the tree is already empty. If it is not empty, it calls the clear function to deallocate all the memory allocated in the tree.

Parameters

none	

Returns

none

Precondition

there may be a tree with memory allocated in it

Postcondition

there not be any memory allocated to the current tree

3.2.2 Member Function Documentation

3.2.2.1 template<typename DataType , typename KeyType > void BSTree< DataType, KeyType >::assignmentHelper (BSTreeNode ∗& *dest,* BSTreeNode ∗ *src*) `[protected]`

assignmentHelper function

This function moves throughout the two trees and assigns the values from the src tree to the dest tree. This function is recursive, so it will call itself until the entire tree is copied. The base case for this call is if the src node is null.

Parameters

dest	- the node that will contain a copy of the src node's data.
src	- the node that contains the informationn that must be copied.

Precondition

two pointers will be passed to this function.

Postcondition

the dest pointer will contain the src pointer's data and recursively call itself to copy the rest of the tree. If the src pointer was null, nothing occured.

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

clear function

This function will remove nodes from the BST using a postorder traversal.

Parameters

none	

Returns

none

Precondition

there may or may not be data in current tree

Postcondition

there will not be data in the current tree and root will be set to NULL

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

clearHelper function

This function checks for the children of each node, deletes them, then deletes the current node. This is done through a recursive call. The base case for this function is if the ptr is equivalent to null.

Parameters

ptr	- a pointer with the address of the current node

Returns

none

Precondition

there may or may not be data in the current BST

Postcondition

there will not be any data in the current BST and all pointers will be set to NULL

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

countHelper function

This function counts the total number of nodes with a recursive call. It stops calling itself if leaf is found, or the current node is null.

Parameters

ptr	- current node in BST
total	- the number of nodes in BST

Returns

int - returns the number of nodes found

Precondition

the number of nodes will be set to zero

Postcondition

the total number of nodes found will be updated

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

getCount function

Counts the total number of nodes in the current tree. This is done by checking to see if the current tree is empty. If it is not empty, it will return the countHelper, which recursively counts the number of nodes.

Parameters

none	

Returns

int - returns 0 if empty. Else, it will return the total number of nodes in current BST.

Precondition

the total number of nodes will not be found

Postcondition

the total number of nodes will be found

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

getHeight function

This function counts the height of the current BST. First this function checks to see if its empty. If not empty, calls returns heightHelper. Else, it returns 0.

Parameters

none	

Returns

int - the height of the current tree

Precondition

the height of the tree will not be output

Postcondition

the height of the tree will be output

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

heightHelper function

this function recursively moves throughout the current tree. The base case is if the current pointer is null or has no children. Otherwise, the recursive call changes depending upon the number of children the node has.

Parameters

ptr	- current node

Returns

int - number of nodes involved in height

Precondition

there is an unknow number of nodes for the greatest height of the tree

Postcondition

the height of the tree will be known

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

insert function

This function inserts the new data into the proper location of the BST. It first checks to see the current tree is empty. If it is empty, it simply sets the root equal to the new data. Otherwise, it calls the insertHelper to place the data in the correct location. If the same data item already exist, the new data item replaces it.

Parameters

newData-Item	- the newest data that must be added to the current binary tree

Returns

none

Precondition

an existing tree will not have the new data item included in it

Postcondition

the current tree will now have the new data item, if the new data item had the same key as an old data item, the the old data item will be replaced

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

insertHelper function

This function inserts the new dataItem into the correct location by checking the key and pointer. If the key is greater, it moves to the right of the current node. If the key is less, it moves to the left of the current node. If it is equivalent, it replaces the current data with the new data. If the pointer reaches a null location, the new data will be inserted into a new node at specified location.

Parameters

ptr	- the current node that is to be compared
newDataItem	- the information that is to be inserted into the binary tree

Returns

none

Precondition

the dataItem will not be inserted into the current BST

Postcondition

the dataItem will be inserted in the correct location of the current BST

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

isEmpty function

This function checks to see if there is any memory currently allocated in the BST

Parameters

none	

Returns

bool - if the root is not null it will return true. It will return false otherwise.

Precondition

it will not be know whether or not there is data

Postcondition

a boolean will be returned which determines whether or not there is data

3.2.2.11 template<typename DataType , typename KeyType > void BSTree< DataType, KeyType >::lessHelper (BSTreeNode ∗ *ptr*, const KeyType & *searchKey*) const [protected]

lessHelper function

This function outputs the keys with a smaller value than the searchKey specified. If the current pointer is not null, a comparison between the current node's key and the given search tree occurs. If current key is less than searchKey, the write helper is called to output the current node and all lesser nodes. Otherwise, a recursive call is used to check the nodes to the left of the current node.

Parameters

ptr	- current node that is needed for comparison
searchKey	- the specified key used to compare

Returns

none

Precondition

lesser keys, if any, will not be output

Postcondition

lesser keys found in current BST will be output

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

overloaded assignement operator

This function takes two intialized trees and assigns the current tree the value of the other tree. This function checks to see if the same tree is assigned itself. If this is true, it returns the same tree; otherwise, it proceeds to clear the current tree and call the assignment helper. The assignment helper then recursively moves throughout the current and "other" tree to copy the values from the "other" tree.

Parameters

other	- the binary tree that is to be copied

Returns

(∗this) - the current tree with the copied values

Precondition

- there will be two initialized trees

Parameters

-	there will be two trees with identical values in them.

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

remove function

This function will remove a specified node. This function will return false if it is empty, or it will call the removeHelper to aide in the locating and removal of said node. To check whether or not the node is actually in the BST at all, the retrieve function is being used.

Parameters

deleteKey	- the key of the matching

Returns

bool - returns true if the specified item was deleted, returns false otherwise

Precondition

there may or may not be a specified node that needs to be deleted from the current tree

Postcondition

there will not be a node that matches the item to be deleted in the current BST.

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

removeHelper function

This function removes the specified node from the BST. It relies on recursive calls throughout the current BST. It has several cases to account for the possible variations of children a node may have.

Parameters

src	- a pointer that has the address to the current pointer
deleteKey	- the key that is used to identify the node that is needed to be deleted

Returns

bool - a boolean statement that determines whether or not the node was removed

Precondition

there may or may not be the node that needs to be deleted in the current BST

Postcondition

the delete key will no longer be in the current BST

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

retrieve function

This function checks to see if there is a data item that currently matches the search key. If the data item is found, it returns true and copies the data item to the search dataItem. Otherwise, it will return false. This function relies upon the retrieve helper function to find the data item.

Parameters

searchKey	- a unique id that corresponds the search dataItem
searchData-Item	- the dataItem that will hold the data of the node with the corresponding key

Returns

bool - returns whether or not the search item was found

Precondition

- data item may or may not be located in the current BST

Postcondition

- data Item will either be found or it will not be found

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

retrieveHelper function

This function moves throughout the BST in search of a dataItem with the matching keys. If key is found, it returns true and updates the search data item. Otherwise, it will check to left and to the right of the current node. If the end of the tree is reached without a solution, false is returned.

Parameters

ptr	- the current node that is going to be compared
searchKey	- the key that will determine if the current node matches, or if the search will continue to the left or to the right
searchData-Item	- the data at the matching node will be assigned to this parameter

Returns

bool - returns whether or not the matching node was found

Precondition

the dataItem with the corresponding searchKey may or may not be in the BST

Postcondition

the matching dataItem will be found, or it will not be found.

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

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

3.2.2.19 template<typename DataType , typename KeyType > void BSTree< DataType, KeyType >::writeHelper (BSTreeNode ∗ *ptr*) const `[protected]`

writeHelper function

This function writes the keys of each dataItem in ascending order. This is done by checking if the current node is null. If it is not null, it will check to the left first to print out the lesser vales. Then it will output the current nodes. Then it will output the nodes to the right because they should be higher in value.

Parameters

ptr	- current node that will check left, be output, then check right

Returns

none

Precondition

no keys will be output

Postcondition

keys will be output in ascending order

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

writeKeys function

This function outputs the keys of each dataItem. This is done in ascending order on one line, and it is seperated by one space between each key. This function calls the write helper function so it can recursively move througout the BST and output the keys in correct order

Parameters

none	

Returns

none

Precondition

the keys will remain hidden from the user

Postcondition

the keys will be output to the screen

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

writeLessThan function

this function outputs all of the values in the current tree with the a key less than the specified searchKey. This functions check to see if current tree is empty, and class the lessHelper to aide in outputting the lesser values.

Parameters

searchKey-	the key that must be compared to each node's key

Returns

none

Precondition

may or may not be values in BST less than searchKey

Postcondition

all values that are less than searchKey are output

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
- show9.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

template<typename DataType, class KeyType> class BSTree< DataType, KeyType >::BSTree-Node

3.3.1 Constructor & Destructor Documentation

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

BSTreeNode constructor

This constructor sets the data item as well as the left and right pointers.

Parameters

nodeData-Item	- the data that is stored inside the node
leftPtr	- a node pointer that points to the node to the left of the current node
rightPtr	- a node pointer that points to the node to the right of the current nodes

Returns

none

Precondition

- there will be an uninitialized node

Postcondition

- there will be an initialized node that holds the information passed to it from the parameters

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 files:

- BSTree.h
- BSTree.cpp

3.4 IndexEntry Struct Reference

Public Member Functions

- int getKey () const
- int getKey () const

Public Attributes

- int acctID
- long recNum

3.4.1 Member Function Documentation

3.4.1.1 int IndexEntry::getKey ( ) const [inline]

3.4.1.2 int IndexEntry::getKey ( ) const [inline]

3.4.2 Member Data Documentation

3.4.2.1 int IndexEntry::acctID

3.4.2.2 long IndexEntry::recNum

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

- database.cpp
- database.cs

3.5 TestData Class Reference

Public Member Functions

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

Private Attributes

- int keyField

3.5.1 Member Function Documentation

3.5.1.1 int **TestData::getKey** ( ) const `[inline]`

3.5.1.2 void **TestData::setKey** (int *newKey*) `[inline]`

3.5.2 Member Data Documentation

3.5.2.1 int **TestData::keyField** `[private]`

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

- test9.cpp

Chapter 4

File Documentation

4.1 BSTree.cpp File Reference

This program is used to build binary search trees, perform basic binary search tree functions, specialized binary search tree functions, and even manipulate database information.

```
#include "BSTree.h"
```

4.1.1 Detailed Description

This program is used to build binary search trees, perform basic binary search tree functions, specialized binary search tree functions, and even manipulate database information.

Author

Henry Huffman

Version

1.1

More specifically, this program has the following basic member functions: default constructor, copy constructor, overloaded assignement operator, deconstructor, insert, retrieve, remove, writeKyes, clear, isEmpty, and showStructor. To specifically see what each of these functions do, please see each of their specific documentation. For specialized functions, it includes all of the following: getCount, getHeight, writeLessThan. Again, if you wish to see what each of these functions do, please see their specific documentation. This program performs basic manipulation of database information by using the functions previously specified in this entry. The program will read from a file, store basic data, and perform basic i/o.

Date

Friday, October 17th, 2014

4.2 BSTree.h File Reference

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

Classes

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

4.3 config.h File Reference

this file enables the testing of specified functions

Defines

- #define LAB9_TEST1 1

  all definitions are set to one;therefore, all functions are enabled
- #define LAB9_TEST2 1
- #define LAB9_TEST3 0

4.3.1 Detailed Description

this file enables the testing of specified functions BSTree class (Lab 9) configuration file. Activate test 'N' by defining the corresponding LAB9_TESTN to have the value 1. Deactive test 'N' by setting the value to 0.

Author

Henry Huffman

Version

1.1

More specifically, this file enables getCount, getHeight, and writeLessThan member functions.

Date

Tuesday, September 30, 2014

4.3.2 Define Documentation

4.3.2.1 #define LAB9_TEST1 1

all definitions are set to one;therefore, all functions are enabled

4.3.2.2 #define LAB9_TEST2 1

4.3.2.3 #define LAB9_TEST3 0

4.4 database.cpp File Reference

This program will use the structs and basic variables given in the shell from the instructor to perform basic data base info manipulation.

```
#include <iostream> #include <fstream> #include "BSTree.-
cpp"
```

Classes

- struct AccountRecord
- struct IndexEntry

Functions

- int main ()

Variables

- const int nameLength = 11
- const long bytesPerRecord = 37

4.4.1 Detailed Description

This program will use the structs and basic variables given in the shell from the instructor to perform basic data base info manipulation.

Author

Henry Huffman

Version

1.1

More specifically, this program reads in from a file, stores the data in the given BST and structs, outputs all the specified data in ascending order, gets input from the user, and outputs data from the given file.

Date

Friday, October 17th, 2014

4.4.2 Function Documentation

4.4.2.1 int main ( )

create a temporary string

take in the searchID

while the end of the data file is not found

place searchID into entry struct

place recNum into entry struct

place current struct into index BST

ignore to get to the end of line

take in the searchID

output all the keys using the write keys member function

take in the specified searchID from the user

check to see if searchID is in index BST

if searchID is found, move to specified record

place acctID from file into specified struct

place firstName from file into specified struct

place lastName from file into specified struct

place the balance from file into specified struct

output the acctID

output firstName

output lastName

output the balance

4.4.3 Variable Documentation

4.4.3.1 const long bytesPerRecord = 37

4.4.3.2 const int nameLength = 11

4.5 database.cs File Reference

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

Classes

- struct AccountRecord
- struct IndexEntry

Functions

- void main ()

Variables

- const int nameLength = 11
- const long bytesPerRecord = 37

4.5.1 Function Documentation

4.5.1.1 void main ( )

4.5.2 Variable Documentation

4.5.2.1 const long bytesPerRecord = 37

4.5.2.2 const int nameLength = 11

4.6 show9.cpp File Reference

4.7 test9.cpp File Reference

`#include <iostream> #include "BSTree.cpp" #include "config.h"`

Classes

- class TestData

Functions

- void print_help ()
- int main ()

4.7.1 Function Documentation

4.7.1.1 int main ( )

4.7.1.2 void print_help ( )