Binary Search Tree

1

Generated by Doxygen 1.8.8

Mon Oct 20 2014 22:59:47

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 AccountRecord Struct Reference 5

3.1.1 Member Data Documentation 5

3.1.1.1 acctID 5

3.1.1.2 balance 5

3.1.1.3 firstName 5

3.1.1.4 lastName 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 6

3.2.1.3 ∼BSTree 7

3.2.2 Member Function Documentation 7

3.2.2.1 clear 7

3.2.2.2 clearHelper 8

3.2.2.3 copyHelp 8

3.2.2.4 countHelper 9

3.2.2.5 getCount 10

3.2.2.6 getHeight 10

3.2.2.7 heightHelper 11

3.2.2.8 insert 11

3.2.2.9 insertHelper 11

3.2.2.10 isEmpty 12

3.2.2.11 operator= 12

3.2.2.12 remove 13

3.2.2.13 removeHelper 13

3.2.2.14 retrieve 14
3.2.2.15 retrieveHelper 14
3.2.2.16 showHelper 15
3.2.2.17 showStructure 15
3.2.2.18 writeKeys 15
3.2.2.19 writeKeysHelper 15
3.2.2.20 writeLessThan 16
3.2.3 Member Data Documentation 16
3.2.3.1 root 16
3.3 BSTree< DataType, KeyType >::BSTreeNode Class Reference 16
3.3.1 Constructor & Destructor Documentation 16
3.3.1.1 BSTreeNode 16
3.3.2 Member Data Documentation 17
3.3.2.1 dataItem 17
3.3.2.2 left 17
3.3.2.3 right 17
3.4 IndexEntry Struct Reference 17
3.4.1 Member Function Documentation 17
3.4.1.1 getKey 17
3.4.2 Member Data Documentation 17
3.4.2.1 acctID 17
3.4.2.2 recNum 17
3.5 TestData Class Reference 18
3.5.1 Member Function Documentation 18
3.5.1.1 getKey 18
3.5.1.2 setKey 18
3.5.2 Member Data Documentation 18
3.5.2.1 keyField 18

4 File Documentation 19

4.1 BSTree.cpp File Reference 19
4.2 BSTree.h File Reference 19
4.3 config.h File Reference 19
4.3.1 Macro Definition Documentation 19
4.3.1.1 LAB9_TEST1 19
4.3.1.2 LAB9_TEST2 20
4.3.1.3 LAB9_TEST3 20
4.4 database.cpp File Reference 20
4.4.1 Function Documentation 20
4.4.1.1 main 20

4.4.2 Variable Documentation 20
4.4.2.1 bytesPerRecord 20
4.4.2.2 nameLength 20
4.5 show9.cpp File Reference 20
4.6 test9.cpp File Reference 20
4.6.1 Function Documentation 21
4.6.1.1 main 21
4.6.1.2 print_help 21

Chapter 1

Class Index

1.1 Class List

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

AccountRecord 5
BSTree< DataType, KeyType > 5
BSTree< DataType, KeyType >::BSTreeNode 16
IndexEntry 17
TestData 18

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

BSTree.cpp 19
BSTree.h 19
config.h 19
database.cpp 20
show9.cpp 20
test9.cpp 20

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[nameLength]

3.1.1.4 char AccountRecord::lastName[nameLength]

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

- database.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 showHelper (BSTreeNode ∗p, int level) const
- void insertHelper (BSTreeNode ∗&location, const DataType &newDataItem)
- bool retrieveHelper (BSTreeNode ∗location, const KeyType &searchKey, DataType &searchDataItem) const
- bool removeHelper (BSTreeNode ∗&location, const KeyType &deleteKey)
- void writeKeysHelper (BSTreeNode ∗location) const
- void clearHelper (BSTreeNode ∗&location)
- int heightHelper (BSTreeNode ∗location) const
- int countHelper (BSTreeNode ∗location) const
- void copyHelp (BSTreeNode ∗&home, BSTreeNode ∗RHS)

Protected Attributes

- BSTreeNode ∗ root

3.2.1 Constructor & Destructor Documentation

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

The default BST constuctor.

Just initializes a new BST tree by setting its root to NULL.

Parameters

None.	

Returns

Constructor.

Precondition

None.

Postcondition

There is a new initialized tree.

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

This is the copy constructor.

Will create a new BST tree with the data from the parameter. Uses the overloaded assignment operator.

Parameters

Another	BST to copy from.

Returns

Constructor.

Precondition

There should be one tree already initialized.

Postcondition

There will be two identical trees.

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

This is the tree destructor.

Deallocates all the memory of the tree by calling the clear function (details of that in function)

Parameters

None	

Returns

Destructor.

Precondition

There should be an initialized tree.

Postcondition

All memory will be deallocated.

3.2.2 Member Function Documentation

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

This is the function that will deallocate all memory of the BST.

Calls its helper then sets the root to NULL.

Parameters

None.	

Returns

void

Precondition

there should an initialized tree.

Postcondition

there will be an empty tree.

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

This is the function that will recursively deallocate all memory.

If the the tree is not empty, goes down all nodes on the left, checks if they have a right child. If they do recalls. If they dont, they get deleted.

Parameters

ExprTree,the	tree to copy to and copy from

Returns

void

Precondition

there should two initialized trees

Postcondition

there will be two identical trees

if there is more to the left recall with child

if leftmost has a right child repeat process with the right child

deallocate

3.2.2.3 template<typename DataType , class KeyType > void BSTree< DataType, KeyType >::copyHelp (BSTreeNode ∗& *home,* BSTreeNode ∗ *RHS*) [protected]

This is the function that will recursively copy two trees to eachother.

Checks if the current Node from the tree to copy from is NULL. If it is, then will end. If the Node is not null will create a new node for the LHS tree with the same data then call itself for the left and right side of that node. Stopping condition is until they reach the end of RHS.

Parameters

Will	take in two BST Node pointers. One from each Tree. Should be corresponding.

Returns

Void.

Precondition

There should two initialized trees.

Postcondition

There will be two identical trees.

Stopping condition, end of tree

Create node

Recall with the other parts of the tree

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

This is the function that will recursively count the number of items in the tree.

Basically adds one and recalls for the left and right every time a node is not NULL. If NULL then just adds 0.

Parameters

BST	node, location.

Returns

Int, the amount of nodes.

Precondition

there should an initialized tree.

Postcondition

tree will be counted.

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

This is the function counts the total amount of nodes in the BST.

calls the helper.

Parameters

None.	

Returns

Int, the amount of nodes.

Precondition

there should an initialized tree.

Postcondition

number of nodes will be returned.

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

This is the function that will get the height of the tree.

Checks if the tree is empty. If it is, returns 0 for height. If it isnt, calls the helper.

Parameters

None.	

Returns

Int, height of the tree.

Precondition

there should an initialized tree.

Postcondition

the height will be returned.

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

This is the function that will recursively check for the height of the tree.

If reached the end returns 0, Otherwise keeps calling itself for the size of both the left and right part.

Parameters

BST	node. Current location.

Returns

Int for the height.

Precondition

there should an initialized tree.

Postcondition

height will be returned

if we reached a null spot, return 0 since nothing is added

get the size of the left and right by recalling and going to the left and right

return which ever is larger plus one since the current node is part of the height

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

This is the function that will insert a new item into the tree.

Inserts new data by calling its helper. Details will be there.

Parameters

The	new data. Could be of different types.

Returns

void

Precondition

there should an initialized tree.

Postcondition

there will be another item in the tree with correct positioning.

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

This is the function that will recursively find where to insert and insert the new data.

This first checks if we have gotten to a NULL pointer in the tree. If it has, then this is the location to insert the new node. If not, compare the data to the current location. If the data is bigger, then recall going right if less then recall going left until the correct positioning.

Parameters

Node	pointer and the data.

Returns

void

Precondition

there should an initialized tree.

Postcondition

there will be a new item.

if reached location, create the new node

if the current location is larger than data go left

if the current location is smaller than data, go right

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

This is the function that checks if the tree is empty.

Checks if the root is null to see if the tree is empty.

Parameters

None.	

Returns

True if empty. False if not.

Precondition

there should an initialized tree.

Postcondition

Nothing changes.

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

This is the overloaded assignment operator.

Will set the tree equal to the tree in the parameter. This is done by calling the copyHelp function (details there.)

Parameters

Another	BST to copy from.

Returns

The copied BST.

Precondition

There should two initialized trees.

Postcondition

There will be two identical trees.

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

This is the function that will look for something and delete it.

Calls its helper.

Parameters

The	thing to delete (keytype)

Returns

True, if found and deleted. False if not found.

Precondition

there should an initialized tree

Postcondition

there will one less node if asked to remove something found.

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

This is the function that will recursively find then delete a node. Then adjusts the tree.

If it has gotten to the end and it has still not found the keytype it ends. Once found checks how many children the node of the item has. If none, just deletes. If it has one then just points the current to the next of next. If two children, find the predeccsor and then overwrites the current data with the predeccesor then calls the removeHelp to delete the original predeccesor Node. If nothing is found but not null recalls with the appropriate side of the tree.

Parameters

BST	node, the current location. What to delete, keytype.

Returns

true if something is deleted. false if not found.

Precondition

there should an initialized tree

Postcondition

something will be deleted and tree will still be in order.

if at the end and not found, return false

if found, delete

if no children

if one child left child, no right

no left, right child

if two children

get predecessor

overwrite the value to delete with the predecessor

delete the node with the value just written to the new location

keep searching

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

This is the function that checks to see if a piece of data is in the BST.

Calls the retrieve helper.

Parameters

The	thing to look for (KeyType) and where to put it (DataType)

Returns

Bool, true if found. False if not.

Precondition

there should an initialized tree

Postcondition

Nothing changes.

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

This is the function that will recursively check for a data item.

Checks if we have gotten to the end. If we have and still not found will return false. If the item in the current location is the same will set it equal to the parameter to copy to. If niether of those are true then checks to see if we are searching for something bigger or smaller than the current and go the corresponding way.

Parameters

BST	node pointer (where to check), the thing to look for (keytype), where to store if found (datatype)

Returns

True if found, false if not.

Precondition

there should an initialized tree

Postcondition

Nothing changes.

if checked the complete tree, return false

if found, return true

if what were are looking for is smaller go left else go right

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

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

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

This is the function that will write the data in the tree in order.

Calls its helper.

Parameters

None.	

Returns

void

Precondition

there should an initialized tree.

Postcondition

data will be printed on the screen.

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

This is the function that will recursively print out all the data in order.

Navigates all the way to the left of the BST, then prints out. Recalls the function with the children to the left of the most left node that has not been printed yet.

Parameters

BST	node. The current location.

Returns

void

Precondition

there should an initialized tree.

Postcondition

data will be printed on the screen.

go all the way to the left

once all the way to the left print out the current

check for the children to the right of the most left node

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

NOT USED

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

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*)

The constructor for a BST node.

Takes the parameters and sets them to its data members

Parameters

The	data item which can be different things, and two BST Node pointers for the left and right children.

Returns

Constructor.

Precondition

There should be an initialized tree.

Postcondition

There will be a new node inside of a BST tree.

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

Public Attributes

- int acctID
- long recNum

3.4.1 Member Function Documentation

3.4.1.1 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 file:

- database.cpp

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

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <sys/time.h>
#include "BSTree.h"
#include "show9.cpp"
```

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

Macros

- #define LAB9_TEST1 1
- #define LAB9_TEST2 1
- #define LAB9_TEST3 0

4.3.1 Macro Definition Documentation

4.3.1.1 #define LAB9_TEST1 1

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.

4.3.1.2 #define LAB9_TEST2 1

4.3.1.3 #define LAB9_TEST3 0

4.4 database.cpp File Reference

```
#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 Function Documentation

4.4.1.1 int main ( )

Gets Size of the file and goes back to the beginning

For the number of lines (size/bytesperrecord) will grab all the info and save the wanted ones inside a BST

Output the account IDs in ascending order.

Clear the status flags for the database file

Read an account ID from the keyboard and output the

if valid ID grab, navigate to the line in the file and then grab all the info and print it if not valid, print error message

get new ID

4.4.2 Variable Documentation

4.4.2.1 const long bytesPerRecord = 37

4.4.2.2 const int nameLength = 11

4.5 show9.cpp File Reference

4.6 test9.cpp File Reference

```
#include <iostream>
```

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

Classes

- class TestData

Functions

- void print_help ()
- int main ()

4.6.1 Function Documentation

4.6.1.1 int main ( )

4.6.1.2 void print_help ( )