Lab 08: BSTree

Generated by Doxygen 1.7.6.1

Wed Oct 30 2013 17:31:14

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 Detailed Description

The struct that has fields for all data types in a database record entry.

3.1.2 Member Data Documentation

3.1.2.1 int AccountRecord::acctID

3.1.2.2 double AccountRecord::balance

3.1.2.3 char AccountRecord::firstName[nameLength]

3.1.2.4 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 clear ()
- void insert (const DataType &newDataItem)
- bool remove (const KeyType &deleteKey)
- bool isEmpty () const
- int getHeight () const
- int getCount () const
- bool retrieve (const KeyType &searchKey, DataType &searchDataItem) const
- void showStructure () const
- void writeKeys () const
- void writeLessThan (const KeyType &searchKey) const

Protected Member Functions

- void clone_sub (BSTreeNode *&currentNode, const BSTreeNode *otherNode)
- void clear_sub (BSTreeNode *&currentNode)
- void insert_sub (BSTreeNode *&currentNode, const DataType &newDataItem, const KeyType &key)
- bool remove_sub (const KeyType &deleteKey, BSTreeNode *&currentNode)
- int getHeight_sub (BSTreeNode *currentNode) const
- int getCount_sub (BSTreeNode *currentNode) const
- bool retrieve_sub (BSTreeNode *currentNode, const KeyType &searchKey, DataType &searchDataItem) const
- void showHelper (BSTreeNode *p, int level) const
- void writeKeys_sub (BSTreeNode *currentNode) const
- void writeLessThan_sub (KeyType &searchKey, BSTreeNode *start, BSTreeNode *predecessor, bool &keysWerePrinted) const

Protected Attributes

- BSTreeNode * root

3.2.1 Detailed Description

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

The class implementations of the Binary Search Tree ADT. This class offers all basic functionality of the Binary Search Tree.

3.2.2 Constructor & Destructor Documentation

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

BSTree

The default constructor for the Binary Search Tree ADT. Constructs an empty tree.

Precondition

1. a tree with the calling identifier has not yet been instantiated

Postcondition

1. an empty tree with the calling identifier will have been created

1. BSTreeNode* root data member is set to NULL

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

BSTree

The copy constructor for the Binary Search Tree ADT. Constructs a clone of the given other tree.

Parameters

other	another binary search tree object of similar types

Precondition

1. a tree with the calling identifier has not yet been created

Postcondition

1. a tree that is a clone of the given other tree has been created

1. BSTree* root data member is set to NULL
2. the overloaded operator= is called

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

~BSTree

The destructor for the Binary Search Tree ADT. Ensures all dynamically allocated memory is returned.

Precondition

1. a tree with the calling identifier has been instantiated

Postcondition

1. *this tree will be destructed properly, returning all dynamically allocated memory

1. calls the clear function to delete all nodes contained in the tree

3.2.3 Member Function Documentation

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

clear

Clears all data from the BSTree.

Precondition

1. a BSTree object has been instantiated

Postcondition

1. all data will be removed from the tree
2. all dynamic memory will be returned
3. the the value of the BSTreeNode* root data member will be NULL

1. if the tree has contents the clear_sub private helper function is called with the BSTreeNode* root data member as a parameter in order to clear the entire tree
2. if the tree is empty, no further action is taken

3.2.3.2 template<typename DataType , typename KeyType > void BSTree< DataType, KeyType >::clear_sub (BSTreeNode *& *currentNode*) [protected]

clear_sub

The private helper function that carries out the clearing of the tree. This fucntion clears a given subtree by clearing the left and right subtrees and then deleting the current node (post-order traversal). Ensures all dynamic memory is returned for the given tree.

Parameters

currentNode	the node currently being considered by the function

Precondition

1. a BSTree has been instantiated
2. the clear_sub function is called using a pointer that points to valid data

Postcondition

1. the BSTree subtrees will be empty
2. the BSTreeNode* currentNode will point to NULL

1. each branch pointer is checked to see if it points to data
2. if the branch pointers point to valid subtrees, these are cleared
3. once any subtrees are cleared, the currentNode is deleted

3.2.3.3 template<typename DataType , typename KeyType > void BSTree< DataType, KeyType >::clone_sub (BSTreeNode *& *currentNode,* const BSTreeNode * *otherNode*) [protected]

clone_sub

The private helper function that carries out a cloning operation by recursively following the nodes of a given BSTree of same type and duplicating its nodes using an in-order traversal.

Parameters

currentNode	the node currently being considered by the function
otherNode	the counterpart node in the other tree that is being cloned

Precondition

1. a BSTree has been, or is being, instantiated
2. the clone_sub function has been called using a node pointer within the tree that points to NULL
3. the BSTreeNode* otherNode must point to valid data
4. other recursive calls previous to this one may have been made

Postcondition

1. BSTreeNode* currentNode will point to an equivalent object to the one poined by BSTreeNode* otherNode
2. recursive calls will be made to continue the process down to the leaves of the tree

1. a new BSTreeNode is created with BSTreeNode* currentNode and given a copy of the dataItem held by other, the left and right pointers are set to NULL
2. checks are made to see if otherNode's left and right pointers are NULL or not
3. the function is called again to clone the other tree's left and right subtrees if possible

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

getCount

Provides public access to find the current count of items stored in the the BSTree.

Returns

int count the count of items in the tree

Precondition

1. a BSTree object has been instantiated

Postcondition

1. the BSTree remains unchanged
2. the current size of the tree is returned

1. the getCount_sub helper is called, starting at the root, to trace the tree and count the nodes
2. the value returned by the helper function is the size of the tree

3.2.3.5 template<typename DataType , typename KeyType > int BSTree< DataType, KeyType >::getCount_sub (BSTreeNode * *currentNode*) const [protected]

getHeight_sub

The private helper function that counts all nodes in the current tree.

Parameters

currentNode	the node currently being considered by the function

Precondition

1. a BSTree has been instantiated
2. this function may have been called previously

Postcondition

1. every valid node in the tree will be visited
2. the count of nodes in the subtree evaluated by the function is returned

1.

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

getHeight

Provides public access to find the current height of the BSTree.

Returns

int height the height of the tree, that is, the number of vertices in the longest chain in the tree

Precondition

1. a BSTree object has been instantiated

Postcondition

1. the BSTree remains unchanged
2. the current height of the tree is returned

1. the getHeight_sub helper is called, starting at the root, to trace the longest chain in the tree
2. the value returned by the helper function is the height of the tree

3.2.3.7 template<typename DataType , typename KeyType > int BSTree< DataType, KeyType >::getHeight_sub (BSTreeNode * *currentNode*) const [protected]

getHeight_sub

The private helper function that determines the height of the tree by counting the vertices of the maximum length chain.

Parameters

currentNode	the node currently being considered by the function
level	the level a node pointed by currentNode would be on

Returns

int height the height of the subtree including currentNode

Precondition

1. a BSTree has been instantiated
2. this function may have been called previously

Postcondition

1. every chain in the train will have been followed in order to count the maximum length chain

1. if BSTreeNode* currentNode points to NULL, the value of level - 1 is returned (base case)
2. otherwise, the function is called to trace its subtrees to continue counting the tree height
3. the subtree heights are compared to find the greatest one

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

insert

Inserts newDataItem into the tree according to the items key. If a data item with the same key already exists in the tree, then it is replaced with newDataItem. Otherwise, a node is created in the appropriate place in the tree to accomodate newDataItem.

Parameters

newDataItem	an object of type DataItem to be inserted into the tree.

Precondition

1. a BSTree object has been instantiated
2. the data type of the templated BSTree supports a getKey() member

Postcondition

1. the tree will contain a BSTreeNode containing newDataItem, appropriately located relative to the pre-existing nodes

1. the key of the data item is obtained
2. the insert_sub helper is called, starting at the root, to locate the correct insertion location for the newDataItem

3.2.3.9 template<typename DataType , typename KeyType > void BSTree< DataType, KeyType >::insert_sub (BSTreeNode *& *currentNode,* const DataType & *newDataItem,* const KeyType & *key*) [protected]

insert_sub

The private helper function that inserts new data into the tree. If a node containing a data item with a key equivalent to the given one, then the data is replaced with the given data. If no match for the given key exists in the tree, a new node is created to accomodate the data.

Parameters

currentNode	the node currently being considered by the function
newDataItem	the data item to be inserted in the tree
key	the key of the item to be inserted

Precondition

1. a BSTree has been instantiated

Postcondition

1. if there is a node with a key equivalent to the given one, then its data is replaced with DataType neDataItem
2. if there is no matching key, a new node is created in the appropriate position of the tree to accommodate newDataItem

1. if the BSTreeNode* points to NULL, a new node is created and given the data of DataType newDataItem

2. otherwise, a check is made to see if it contains an equivalent key to the given one, if so, the data replacement operation is conducted
3. if the keys still don't match, checks to see which subtree the data with the given key belongs in, and the helper function is called to insert the data in the appropriate subtree

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

isEmtpy

Reports the state of the tree, specifically, returns true if the BSTree is empty, and false if the tree has any contents.

Returns

bool empty the truth value of the BSTree containing no nodes

Precondition

1. a BSTree object has been instantiated

Postcondition

1. the BSTree remains unchanged
2. the state of the tree is indicated via the return of a boolean flag

1. the remove_sub helper is called, starting at the root, to locate the given key, if possible, and remove the node
2. if a removal occured true is returned, otherwise false is returned

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

operator=

The overloaded assignment operator. Assigns a clone of BSTree other to ∗this.

Parameters

other	another binary search tree of similar type

Returns

∗this

Precondition

1. a tree with the calling identifier has been or is being created

Postcondition

1. a tree that is a clone of the given other tree has been created
2. a reference to the new ∗this is returned

1. a check to see if ∗this is being assigned to itself is performed
2. otherwise ∗this is cleared
3. if BSTree other is not empty, the clone_sub private helper function is called to carry out the cloning process
4. if BSTree other is empty, no further action is taken
5. a reference to ∗this is returned

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

remove

Removes a node with the given key from the tree. Returns the success of the removal operation.

Parameters

deleteKey	the key used to locate the item to be removed

Precondition

1. a BSTree object has been instantiated
2. the data type of the templated BSTree supports a getKey() member

Postcondition

1. if it exists, the node containing the data item with the given key will be removed
2. the success of the operation is returned to the calling function

1. the remove_sub helper is called, starting at the root, to locate the given key, if possible, and remove the node

2. if a removal occured true is returned, otherwise false is returned

3.2.3.13 template<typename DataType , typename KeyType > bool BSTree< DataType, KeyType >::remove_sub (const KeyType & *deleteKey,* BSTreeNode *& *currentNode*) [protected]

remove_sub

The private helper function used to delete data from the tree. In general, this function performs similar to a remove function in a linked list. In the case that a node to be removed has two children, the node is replaced with its "in-order predecessor."

Parameters

deleteKey	fad
currentNode	the pointer that points to the node currently being considered. Note: this parameter is passed by reference, so it is actually the pointer belonging to the node's predecessor (or the root data member).

Returns

bool removed a flag to indicate whether or not a node was located and removed (true for removal, false for failure to remove)

Precondition

1. a BSTree has been instantiated
2. BSTreeNod* currentNode points to the current location of the tree being considered currently
3. recursive calls to this function may have been made previously

Postcondition

1. if a node containing data with a key equivalent to KeyType searchKey is found, it is removed in an appropriate manner and true is returned
2. otherwise, nothing occurs and false is returned
3. there are no guarantees as to the relative structure of the tree other than the tree will still fit the definition of a binary search tree

1. a search for a key equivalent to KeyType searchKey is conducted
2. if the function is called on a NULL pointer, then the search has failed, no removal can be performed, and false is returned
3. if the key matches a leaf, then the leaf is simply deleted

4. if the key matches a node with one child, then the tree is simply re-linked and the BSTreeNode* currentNode is linked to the appropriate child, and the excluded node is deleted
5. if the key matches a node with 2 children, then the "in-order predecessor" of the node is found, and the original node's data is replaced with that of the predecessor, and the now reduntant predecessor node is then deleted

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

retrieve

Provides public access to find the item of the given key stored in the tree. The success of the operation is returned, while searchDataItem is modified by reference. If the operation fails, searchDataItem is left unchanged.

Parameters

searchKey	the key corresponding to the sought item
searchDataItem	the reference variable used to store the sought item if found

Returns

bool found the sucess in finding the sought item with the given key

Precondition

1. a BSTree object has been instantiated

Postcondition

1. the BSTree remains unchanged
2. the data item with the given key is passed back by reference if found
3. the truth value of whether or not the sought item was found

1. the retrieve_sub helper is called to search the tree for the item with the given key
2. if the given key is found, the DataType searchDataItem passed by reference will be given the value of the item with the sought key
3. if the item with the given key was found, true is returned, otherwise false is returned

3.2.3.15 template<typename DataType , typename KeyType > bool BSTree< DataType, KeyType >::retrieve_sub (BSTreeNode * *currentNode,* const KeyType & *searchKey,* DataType & *searchDataItem*) const [protected]

retrieve_sub

The private helper function that counts all nodes in the current tree.

Parameters

currentNode	the node currently being considered by the function

Precondition

1. a BSTree has been instantiated
2. this function may have been called previously

Postcondition

1. if the item with the given key is found, it is passed back by reference in the DataType searchDataItem, otherwise it is left unmodified and false is returned
2. the BSTree remains unchanged

1. if the item with the given key is found, it is passed back by reference and true is returned to indicate that the item was found
2. if the function is called with a NULL pointer, the item was not found, DataType searchDataItem is not modified, and true is returned

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

showHelper

The private helper function that works to output the contents of the tree to the screen.

Parameters

p	a pointer to the node currently being considered
level	the level of the nodes to be output by this function call

Precondition

1. a BSTree has been instantiated
2. this function may have been called previously
3. the DataType must support the << operator

Postcondition

1. the tree will remain unchanged
2. the contents of the given subtree will be displayed on the screen

1.

3.2.3.17 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.

PROVIDED BY THE LAB MANUAL PACKAGE

Precondition

1. a BSTree object has been instantiated

Postcondition

1. the BSTree remains unchanged
2. the structure of the tree is displayed on the screen

1. the showHelper function is called to display the tree, starting from the root

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

writeKeys

Provides public access to have the keys currently contained in the tree listed in increasing order on the screen. The DataType used must support the << operator.

Precondition

1. a BSTree object has been instantiated

Postcondition

1. the BSTree remains unchanged
2. the keys of the data items will be listed in increasing order on the screen, separated by a space

1. the writeKeys_sub helper function is called to carry out the process of displaying the keys, starting from the root
2. if the tree is empy, it is reported

3.2.3.19 template<typename DataType , typename KeyType > void BSTree< DataType, KeyType >::writeKeys_sub (BSTreeNode * *currentNode*) const [protected]

writeKeysSub

The private helper function that works to output all keys in ascending order for a given subtree.

Parameters

currentNode	a pointer to the node currently being considered

Precondition

1. a BSTree has been instantiated
2. BSTreeNode* currentNode must point to valid data
3. this function may have been called previously
4. the DataType must support the << operator

Postcondition

1. the tree will remain unchanged
2. the keys of the given subtree will be listed on the screen

1. a check to ensure the given pointer is not null is performed
2. all keys in the left subtree of the tree currently being considered are listed first
3. the key of the current node is listed
4. all keys of the right subtree of the tree currently being considered are then listed

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

writeLessThan

Provides public access to have the keys currently contained in the tree listed in increasing order up to the given bound on the screen. The DataType used must support the << operator.

Parameters

searchKey	the upper bound of the keys to be listed. If this key is in the list, it is listed, otherwise the key nearest the given parameter is the larges key listed

Precondition

1. a BSTree object has been instantiated

Postcondition

1. the BSTree remains unchanged
2. the keys of the data items will be listed in increasing order on the screen, up to the upper bound, separated by a space

1. the writeLessThan_sub helper is called to search the tree for the item with the given key, or the nearest one that is less than the given key, starting at the root
2. all keys less than and including the given bound KeyType searchKey are listed if they are present in the tree
3. if no such keys exist, this is reported.

3.2.3.21 template<typename DataType , typename KeyType > void BSTree< DataType, KeyType >::writeLessThan_sub (KeyType & *searchKey,* BSTreeNode * *start,* BSTreeNode * *predecessor,* bool & *keysWerePrinted*) const [protected]

writeLessThan_sub

Lists all keys that are less than or equal to the given key (searchKey) on the screen.

Parameters

searchKey	the key given to indicate the upper (inclusive) bound for the keys to be printed
start	a pointer for the node to start at
predecessor	a pointer to the start node's predecessor, call with NULL if such a node does not exist
keysWerePrinted	a flag to be passed through all calls to indicate wether or not keys less than the search key have been found

Precondition

1. a BSTree has been instantiated
2. BSTreeNode* start must point to valid data
3. BSTreeNode* predecessor must point to either a parent of start or contain the value NULL
4. previous recursive calls may have been made

Postcondition

1. the tree will remain unchanged
2. once all recursive calls have resolved, all keys less than KeyType searchKey will be written in ascending order to the screen

1. the currentNode "cursor" is moved right as far as possible
2. with each move, the previous node and all nodes with keys less than it are listed
3. once the "cursor" has moved too far to the right, it is then moved left in an attempt to find more nodes with keys meeting the search criteria
4. a recursive call is made to repeat the process
5. the task completes when a call is made and no keys could be successfully printed (base case)
6. if no keys can be printed, this is indicated

3.2.4 Member Data Documentation

3.2.4.1 template<typename DataType, typename 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

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

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

The default constructor for the BSTreeNode inner class. This constructor is parameterized. Initializes the BSTreeNode members to the values of the given parameters.

Parameters

nodeDataItem	the data item the new BSTreeNode will contain
leftPtr	a pointer to a left child
rightPtr	a pointer to a right child

Precondition

1. a BSTree object has been instantiated for the BSTreeNode to exist in

Postcondition

1. a BSTreeNode has been created with data members equivalent to the given parameter data

1. simply sets all data members of the node to the given data

3.3.2 Member Data Documentation

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

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

3.3.2.3 template<typename DataType, typename 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 Detailed Description

The struct that pairs the account ID number and the record number that is used by the program

3.4.2 Member Function Documentation

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

getKey

Returns the key of the IndexEntry struct. This is the account number for a database entry

Precondition

1. the struct should contain valid data

Postcondition

1. an account number corresponding to a record number is returned

3.4.3 Member Data Documentation

3.4.3.1 int IndexEntry::acctID

3.4.3.2 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

Class implementations declarations for the linked implementation of the Binary Search Tree ADT -- including the recursive helpers of the public member functions.

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

4.1.1 Detailed Description

Class implementations declarations for the linked implementation of the Binary Search Tree ADT -- including the recursive helpers of the public member functions.

Author

Terence Henriod

Laboratory 8

Version

Original Code 1.00 (10/29/2013) - T. Henriod

4.2 BSTree.h File Reference

Class declarations for the linked implementation of the Binary Search Tree ADT -- including the recursive helpers of the public member functions.

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

Classes

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

Variables

- const bool LEFT = true
- const bool RIGHT = false

4.2.1 Detailed Description

Class declarations for the linked implementation of the Binary Search Tree ADT -- including the recursive helpers of the public member functions.

Author

Terence Henriod

Labaratory 9

Version

Original Code 1.00 (10/29/2013) - T. Henriod

4.2.2 Variable Documentation

4.2.2.1 const bool LEFT = true

4.2.2.2 const bool RIGHT = false

4.3 config.h File Reference

Defines

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

4.3.1 Define 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

(Shell) Indexed accounts database program. This program should be used with the database file accounts.dat. Builds a binary search tree index for the account records in the text file accounts.dat.

```
#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

(Shell) Indexed accounts database program. This program should be used with the database file accounts.dat. Builds a binary search tree index for the account records in the text file accounts.dat.

Author

Terence Henriod

Laboratory 11, In-lab exercise 1

Version

Original Code 1.00 (10/29/2013) - Modified by T. Henriod

4.4.2 Function Documentation

4.4.2.1 int main ( )

4.4.3 Variable Documentation

4.4.3.1 const long bytesPerRecord = 37

4.4.3.2 const int nameLength = 11

4.5 test9.cpp File Reference

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

Classes

- class TestData

Functions

- void print_help ()
- int main ()

4.5.1 Function Documentation

4.5.1.1 int main ( )

4.5.1.2 void print_help ( )