Project3:WordCount Reference Manual

Generated by Doxygen 1.2.17

Wed Jul 24 18:23:14 2002

Contents

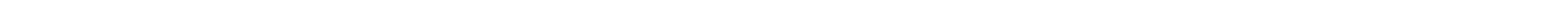
Chapter 1

Project3:WordCount Namespace Index

1.1 Project3:WordCount Namespace List

Here is a list of all namespaces with brief descriptions:

std . . . . ??



Chapter 2

Project3:WordCount Hierarchical Index

2.1 Project3:WordCount Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- BSTNode< DataType > . . . . **??**
- DictionaryADT< KeyType, ValueType > . . . . **??**
  - BinarySearchTree< KeyType, ValueType > . . . . **??**

Chapter 3

Project3:WordCount Compound Index

3.1 Project3:WordCount Compound List

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

BinarySearchTree< KeyType, ValueType > (The BinarySearchTree class) . . ??
BSTNode< DataType > (The Binary Search Tree node struct) . . . . ??
DictionaryADT< KeyType, ValueType > (This abstract class gives the basic interface of the DictionaryADT) . . . . ??

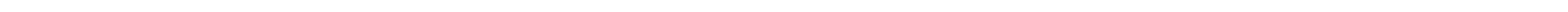
Chapter 4

Project3:WordCount File Index

4.1 Project3:WordCount File List

Here is a list of all files with brief descriptions:

BinarySearchTree.cc (Implementation of a templated BinarySearchTree class which maps keys to values (duplicated keys are not allowed)) . ??
BinarySearchTree.hh (Specification of a templated BinarySearchTree class which maps keys to values (duplicated keys are not allowed)) . . . . ??
BSTInst.cc (Instantiation file for the templated BST classes) . . . . ??
BSTNode.cc (Implementation of a templated BSTNode class which represents generic nodes in a Binary Search Tree) . . . . ??
BSTNode.hh (Specification of a templated BSTNode containing Key/Value pairs) . . . . ??
dictionary-test.cc (Simple program for testing BST functionality (note that you can use it to test your AVL/Splay trees as well!)) . . . . ??
DictionaryADT.hh (Specification of a templated DictionaryADT class which maps keys to values (duplicated keys are not allowed)) . . . . ??



Chapter 5

Project3:WordCount Namespace Documentation

5.1 std Namespace Reference

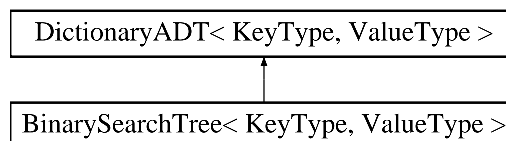
Chapter 6

Project3:WordCount Class Documentation

6.1 BinarySearchTree< KeyType, ValueType > Class Template Reference

```
#include <BinarySearchTree.hh>
```

Inheritance diagram for BinarySearchTree< KeyType, ValueType >::



6.1.1 Detailed Description

template<typename KeyType, typename ValueType> class BinarySearchTree< KeyType, ValueType >

The BinarySearchTree class.

Definition at line 32 of file BinarySearchTree.hh.

Public Types

- typedef DictionaryADT< KeyType, ValueType >::Entry Entry

  A local typedef of the DictionaryADT::Entry mainly kept here for for documentation purposes.

- typedef DictionaryADT< KeyType, ValueType >::EntryList EntryList

  A local typedef of the DictionaryADT::EntryList mainly kept here for for documentation purposes.

- typedef BSTNode< Entry > Node

  Typedef for the type of ndoe used by this BST.

Public Methods

- BinarySearchTree (void)

  (1) Default constructor.

- BinarySearchTree (const BinarySearchTree &other)

  (2) Copy constructor.

- BinarySearchTree & operator= (const BinarySearchTree &other)

  (3) Assign Op.

- virtual ~BinarySearchTree (void)

  (4) Destructor.

- virtual int getSize (void) const

  Returns the number of elements in the BST.

- virtual bool find (const KeyType &key, ValueType &value) const

  Given a key, 'returns' (via a reference parameter) the value associated with it in this BST.

- virtual int getEntryList (EntryList &entries) const

  "Returns" a list of all the Key/Value pairs in the tree.

- virtual std::ostream & print (std::ostream &out=std::cerr) const

  Pretty-Print out a view of this Dictionary for debugging purposes.

- virtual bool insert (const KeyType &newKey, const ValueType &newValue)

  Inserts a new key/value pair into the BST.

- virtual bool changeValue (const KeyType &oldKey, const ValueType &new-Value)

  Changes the value associated with an existing key in the BST (oldKey) to a new value.

- virtual bool insert (const Entry &entry)

  Same as the the other `insert` *function, but takes an* `Entry` *rather than seperate key and value arguments.*

- virtual bool changeValue (const Entry &entry)

  Same as the the other `ChangeValue` *function, but takes an* `Entry` *rather than seperate key and value argument.*

Protected Methods

- virtual Node ∗ findNode (const KeyType &key) const

  Finds the node associated with a given key.

- virtual int recursiveGetEntryList (EntryList &entries, Node ∗pCurrNode) const

  Recursively copies the key/value pairs in the tree rooted at pCurrNode into the Entry-List.

- virtual Node ∗ cloneTree (const Node ∗const pRoot)

  Recursively copy the tree rooted at pRoot, returning a pointer to the newly copied subtree.

- virtual void destroyTree (Node ∗const pRoot)

  Deallocates memory for the tree rooted at pRoot.

Protected Attributes

- Node ∗ m_pRoot

  The root of the BST.

- int m_numElts

  The number of elements in the BST.

6.1.2 Member Typedef Documentation

6.1.2.1 template<typename KeyType, typename ValueType> template BinarySearchTree< KeyType, ValueType >::Entry

A local typedef of the DictionaryADT::Entry mainly kept here for for documentation purposes.

Used to represent Key/Value pairs.

Reimplemented from DictionaryADT< KeyType, ValueType >.

Definition at line 49 of file BinarySearchTree.hh.

Referenced by BinarySearchTree< KeyType, ValueType >::changeValue(), and BinarySearchTree< KeyType, ValueType >::insert().

6.1.2.2 template<typename KeyType, typename ValueType> template BinarySearchTree< KeyType, ValueType >::EntryList

A local typedef of the DictionaryADT::EntryList mainly kept here for for documentation purposes.

Used by `getEntryList()`.

Reimplemented from DictionaryADT< KeyType, ValueType >.

Definition at line 53 of file BinarySearchTree.hh.

6.1.2.3 template<typename KeyType, typename ValueType> template BinarySearchTree< KeyType, ValueType >::Node

Typedef for the type of ndoe used by this BST.

Useful layer of abstraction that allows a bit more flexibility in changing underly class for each node. (For instance, if you wanted to template the node on an additional parameter, you only have to change this typedef).

Definition at line 59 of file BinarySearchTree.hh.

Referenced by BinarySearchTree< KeyType, ValueType >::insert().

6.1.3 Constructor & Destructor Documentation

6.1.3.1 template<typename KeyType, typename ValueType> BinarySearchTree< KeyType, ValueType >::BinarySearchTree (void)

(1) Default constructor.

Constructs an empty tree.

Definition at line 26 of file BinarySearchTree.cc.

6.1.3.2 template<typename KeyType, typename ValueType> BinarySearchTree< KeyType, ValueType >::BinarySearchTree (const BinarySearchTree< KeyType, ValueType > & *other*)

(2) Copy constructor.

Constructs a new tree that is a copy of the given tree.

Parameters:

other The tree to copy.

Definition at line 36 of file BinarySearchTree.cc.

References BinarySearchTree< KeyType, ValueType >::cloneTree(), BinarySearchTree< KeyType, ValueType >::m_numElts, and BinarySearchTree< KeyType, ValueType >::m_pRoot.

6.1.3.3 template<typename KeyType, typename ValueType> BinarySearchTree< KeyType, ValueType >::~BinarySearchTree (void) `[virtual]`

(4) Destructor.

Note that the destructor is declared virtual in this inheritance tree. Why do you think this is necessary?

Definition at line 63 of file BinarySearchTree.cc.

References BinarySearchTree< KeyType, ValueType >::destroyTree(), and BinarySearchTree< KeyType, ValueType >::m_pRoot.

6.1.4 Member Function Documentation

6.1.4.1 template<typename KeyType, typename ValueType> bool BinarySearchTree< KeyType, ValueType >::changeValue (const Entry & *entry*) `[virtual]`

Same as the the other `ChangeValue` function, but takes an `Entry` rather than seperate key and value argument.

Returns :

`true` if the data was changed, `false` otherwise.

Parameters:
entry The entry representing the key/value information to be changed in the BST.

Implements DictionaryADT< KeyType, ValueType >.

Definition at line 214 of file BinarySearchTree.cc.

References BinarySearchTree< KeyType, ValueType >::findNode(), and BSTNode< Entry >::m_data.

6.1.4.2 template<typename KeyType, typename ValueType> bool BinarySearchTree< KeyType, ValueType >::changeValue (const KeyType & *oldKey*, const ValueType & *newValue*) `[virtual]`

Changes the value associated with an existing key in the BST (oldKey) to a new value.

Assumes 'oldKey' exists in the BST.

Returns :
`true` if the data was changed, `false` otherwise.

Parameters:
oldKey The key whose value will be changed.
newValue The new value to be associated with `oldKey`.

Implements DictionaryADT< KeyType, ValueType >.

Definition at line 145 of file BinarySearchTree.cc.

References BinarySearchTree< KeyType, ValueType >::Entry.

Referenced by main().

6.1.4.3 template<typename KeyType, typename ValueType> BinarySearchTree< KeyType, ValueType >::Node * BinarySearchTree< KeyType, ValueType >::cloneTree (const Node *const *pRoot*) `[protected, virtual]`

Recursively copy the tree rooted at pRoot, returning a pointer to the newly copied subtree.

Does *not* deallocate any memory.

Returns :
A pointer to the newly allocated clone of the subtree.

Parameters:
pRoot The root of the subtree to clone

Definition at line 315 of file BinarySearchTree.cc.

References BSTNode< Entry >::clone(), BSTNode< Entry >::m_pLeft, BSTNode< DataType >::m_pParent, and BSTNode< Entry >::m_pRight.

Referenced by BinarySearchTree< KeyType, ValueType >::BinarySearchTree(), and BinarySearchTree< KeyType, ValueType >::operator=().

6.1.4.4 template<typename KeyType, typename ValueType> void BinarySearchTree< KeyType, ValueType >::destroyTree (Node ∗const *pRoot*) [protected, virtual]

Deallocates memory for the tree rooted at pRoot.

Parameters:

pRoot The root of the tree to desctroy.

Definition at line 348 of file BinarySearchTree.cc.

References BSTNode< Entry >::m_pLeft, and BSTNode< Entry >::m_pRight.

Referenced by BinarySearchTree< KeyType, ValueType >::operator=(), and BinarySearchTree< KeyType, ValueType >::∼BinarySearchTree().

6.1.4.5 template<typename KeyType, typename ValueType> bool BinarySearchTree< KeyType, ValueType >::find (const KeyType & *key*, ValueType & *value*) const [virtual]

Given a key, 'returns' (via a reference parameter) the value associated with it in this BST.

Returns :

`true` if the key is in the BST, `false` otherwise. If the key is not in the BST, the value of `value` is undefined.

Parameters:

key The key to search for.

value output parameter to hold a copy of the value for the given key.

Implements DictionaryADT< KeyType, ValueType >.

Definition at line 80 of file BinarySearchTree.cc.

References BinarySearchTree< KeyType, ValueType >::findNode(), and BSTNode< Entry >::m_data.

Referenced by main().

6.1.4.6 template<typename KeyType, typename ValueType> BinarySearchTree< KeyType, ValueType >::Node * BinarySearchTree< KeyType, ValueType >::findNode (const KeyType & *key*) const [protected, virtual]

Finds the node associated with a given key.

If the key is not in the tree, returns `NULL` if the tree is empty; otherwise, returns the parent of where <cdoe>key should be.

Returns :

`NULL` if the tree is empty. The node represeting the key if it exist, or the "parent" of where the node with the `key` should have been.

Parameters:

key The key to search for.

Definition at line 237 of file BinarySearchTree.cc.

References BSTNode< Entry >::m_data, BSTNode< Entry >::m_pLeft, BSTNode< Entry >::m_pRight, and BinarySearchTree< KeyType, ValueType >::m_pRoot.

Referenced by BinarySearchTree< KeyType, ValueType >::changeValue(), BinarySearchTree< KeyType, ValueType >::find(), and BinarySearchTree< KeyType, ValueType >::insert().

6.1.4.7 template<typename KeyType, typename ValueType> int BinarySearchTree< KeyType, ValueType >::getEntryList (EntryList & *entries*) const [virtual]

"Returns" a list of all the Key/Value pairs in the tree.

The values will be in *sorted order*, since this is a BST and the list is generated using an inorder traversal.

Returns :

an int specifying the number of elements in the arrays. -1 if there was an error.

Parameters:

entries output parameter that holds the list of key/value pairs in the BST. The original contents of the list is erased.

Implements DictionaryADT< KeyType, ValueType >.

Definition at line 123 of file BinarySearchTree.cc.

References BinarySearchTree< KeyType, ValueType >::getSize(), BinarySearchTree< KeyType, ValueType >::m_pRoot, and BinarySearchTree< KeyType, ValueType >::recursiveGetEntryList().

Referenced by main().

6.1.4.8 template<typename KeyType, typename ValueType> int BinarySearchTree< KeyType, ValueType >::getSize (void) const [virtual]

Returns the number of elements in the BST.

Returns :
the number of elements in the BST.

Implements DictionaryADT< KeyType, ValueType >.

Definition at line 72 of file BinarySearchTree.cc.

References BinarySearchTree< KeyType, ValueType >::m_numElts.

Referenced by BinarySearchTree< KeyType, ValueType >::getEntryList(), and main().

6.1.4.9 template<typename KeyType, typename ValueType> bool BinarySearchTree< KeyType, ValueType >::insert (const Entry & *entry*) [virtual]

Same as the the other `insert` function, but takes an `Entry` rather than seperate key and value arguments.

Returns :
`true` if this key did not previously exist in the tree, `false` otherwise.

Parameters:
entry The new key/value pair to be inserted into the BST.

Implements DictionaryADT< KeyType, ValueType >.

Definition at line 166 of file BinarySearchTree.cc.

References BinarySearchTree< KeyType, ValueType >::findNode(), BSTNode< Entry >::m_data, BinarySearchTree< KeyType, ValueType >::m_numElts, BSTNode< Entry >::m_pLeft, BSTNode< Entry >::m_pRight, BinarySearchTree< KeyType, ValueType >::m_pRoot, and BinarySearchTree< KeyType, ValueType >::Node.

6.1.4.10 template<typename KeyType, typename ValueType> bool BinarySearchTree< KeyType, ValueType >::insert (const KeyType & *newKey*, const ValueType & *newValue*) [virtual]

Inserts a new key/value pair into the BST.

If the key previously existed in the BST, will overwrite the old value with the new value.

Returns :
`true` if this key did not previously exist in the tree, `false` otherwise.

Parameters:
newKey The new key with which `newValue` will be associated.

newValue the value to be associated with `newKey`.

Implements DictionaryADT< KeyType, ValueType >.

Definition at line 155 of file BinarySearchTree.cc.

References BinarySearchTree< KeyType, ValueType >::Entry.

Referenced by main().

6.1.4.11 template<typename KeyType, typename ValueType> BinarySearchTree< KeyType, ValueType > & BinarySearchTree< KeyType, ValueType >::operator= (const BinarySearchTree< KeyType, ValueType > & *other*)

(3) Assign Op.

Copies the contents of another BST to this BST.

Returns :
returns a reference to *this* BST. So stuff like (a=b).insert(3,4) inserts int BST a. This is the standard C/C++ semantics for assignment.

Parameters:
other The tree to copy.

Definition at line 48 of file BinarySearchTree.cc.

References BinarySearchTree< KeyType, ValueType >::cloneTree(), BinarySearchTree< KeyType, ValueType >::destroyTree(), BinarySearchTree< KeyType, ValueType >::m_numElts, and BinarySearchTree< KeyType, ValueType >::m_pRoot.

6.1.4.12 template<typename KeyType, typename ValueType> ostream & BinarySearchTree< KeyType, ValueType >::print (std::ostream & *out* = std::cerr) const `[virtual]`

Pretty-Print out a view of this Dictionary for debugging purposes.

This code requires operator<< for both KeyType and ValueType, so you can remove this if you're not debugging. You can "turn off" printing by compiling in "release mode" (ie, by defining "NDEBUG")

Returns :

The ostream used by the print function. This is mainly a convienience thing.

Parameters:

out The output stream to pretty-print the Dictionary to. This funciton defaults to cerr.

Implements DictionaryADT< KeyType, ValueType >.

Definition at line 102 of file BinarySearchTree.cc.

References BinarySearchTree< KeyType, ValueType >::m_pRoot, and BSTNode< Entry >::print().

Referenced by main().

6.1.4.13 template<typename KeyType, typename ValueType> int BinarySearchTree< KeyType, ValueType >::recursiveGetEntryList (EntryList & *entries*, Node * *pCurrNode*) const `[protected, virtual]`

Recursively copies the key/value pairs in the tree rooted at pCurrNode into the EntryList.

The copying is done via an in-order traversal of the subtrees rooted at pCurrNode (thus, the data in the arrays will be ordered by key value). Does nothing if pCurrNode == `NULL`.

Returns :

The number of valid items in the list.

Parameters:

entries The EntryList to populate with the key/value pairs

pCurrNode The current node in the inorder traversal

Definition at line 276 of file BinarySearchTree.cc.

References BSTNode< Entry >::isLeaf(), BSTNode< Entry >::m_data, BSTNode< Entry >::m_pLeft, and BSTNode< Entry >::m_pRight.

Referenced by BinarySearchTree< KeyType, ValueType >::getEntryList().

6.1.5 Member Data Documentation

6.1.5.1 template<typename KeyType, typename ValueType> int BinarySearchTree< KeyType, ValueType >::m_numElts [protected]

The number of elements in the BST.

Definition at line 291 of file BinarySearchTree.hh.

Referenced by BinarySearchTree< KeyType, ValueType >::BinarySearchTree(), BinarySearchTree< KeyType, ValueType >::getSize(), BinarySearchTree< KeyType, ValueType >::insert(), and BinarySearchTree< KeyType, ValueType >::operator=().

6.1.5.2 template<typename KeyType, typename ValueType> Node* BinarySearchTree< KeyType, ValueType >::m_pRoot [protected]

The root of the BST.

What is a "mutable" data member? It's a data member that can be modified >>even if the instance of this class is declared const<<! This has one very important effect: This data member can be modifed by any class method, even if the method is declared 'const'! The reason the root pointer is declared mutable will become important when we look at SplayTree::Find() and SplayTree::Splay()

Definition at line 288 of file BinarySearchTree.hh.

Referenced by BinarySearchTree< KeyType, ValueType >::BinarySearchTree(), BinarySearchTree< KeyType, ValueType >::findNode(), BinarySearchTree< KeyType, ValueType >::getEntryList(), BinarySearchTree< KeyType, ValueType >::insert(), BinarySearchTree< KeyType, ValueType >::operator=(), BinarySearchTree< KeyType, ValueType >::print(), and BinarySearchTree< KeyType, ValueType >::~BinarySearchTree().

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

- BinarySearchTree.hh
- BinarySearchTree.cc
- BSTInst.cc

6.2 BSTNode< DataType > Struct Template Reference

```
#include <BSTNode.hh>
```

6.2.1 Detailed Description

template<typename DataType> struct BSTNode< DataType >

The Binary Search Tree node struct.

Contains the data for each node and also performs some important calculations on nodes to analyze tree structure.

Definition at line 49 of file BSTNode.hh.

Public Methods

- BSTNode (const DataType &data, BSTNode *pParent=NULL, BSTNode *pLeft=NULL, BSTNode *pRight=NULL)

  Construct a node by copying the given data members.

- virtual BSTNode * clone (void) const

  Copy the current instance into a new BSTNode.

- virtual bool hasParent (void) const

  *True iff this node has a parent (i.e., if this node is *not* the root).*

- virtual bool hasGrandparent (void) const

  True iff this node's parent has a parent.

- virtual bool isLeftChild (void) const

  Tests if this node is the left child of its parent.

- virtual bool isRightChild (void) const

  Tests if this node is the right child of its parent.

- virtual bool isLeaf (void) const

  Tests of this node is a leaf (has no childern).

- virtual int getNumChildren (void) const

  Gets the number of children this node has.

- virtual bool isZigZig (void) const

Test if this node has a "Zig-Zig" relation to the root of the tree.

- virtual bool isZigZag (void) const

  Test if this node has a "Zig-Zag" relation to the root of the tree.

- virtual void print (std::ostream &out=std::cout, std::string filler="") const

  Pretty-Print this node and its subtree.

- virtual BSTNode * rotateRight (void)

  Rotate this node with its right child, returning a pointer to node that is now in its place.

- virtual BSTNode * rotateLeft (void)

  Rotate this node with its left child, returning a pointer to node that is now in its place.

Public Attributes

- DataType m_data

  The data contained by this node.

- BSTNode * m_pParent

  pointer to this node's parent.

- BSTNode * m_pLeft

  pointer to this node's left child.

- BSTNode * m_pRight

  pointer to this node's right child.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 template<typename DataType> BSTNode< DataType >::BSTNode (const DataType & *data*, BSTNode< DataType > * *pParent* = NULL, BSTNode< DataType > * *pLeft* = NULL, BSTNode< DataType > * *pRight* = NULL)

Construct a node by copying the given data members.

Parameters:

data data item to initalize the node to.

pParent Pointer to parent node. Defaults to `NULL`.

pLeft Pointer to left child node. Defaults to `NULL`.

pRight Pointer to right child node. Defaults to `NULL`.

Definition at line 20 of file BSTNode.cc.

Referenced by BSTNode< DataType >::clone().

6.2.3 Member Function Documentation

6.2.3.1 template<typename DataType> BSTNode< DataType > * BSTNode< DataType >::clone (void) const `[virtual]`

Copy the current instance into a new BSTNode.

The client is responsible for deleting this newly allocated BSTNode

Returns :
a dynamically allocated clone of the current node.

Definition at line 40 of file BSTNode.cc.

References BSTNode< DataType >::BSTNode(), and BSTNode< DataType >::m_data.

6.2.3.2 template<typename DataType> int BSTNode< DataType >::getNumChildren (void) const `[virtual]`

Gets the number of children this node has.

As this is the node for a binary tree, the value will be either 0, 1, or 2.

Returns :
the number of children this node has.

Definition at line 88 of file BSTNode.cc.

References BSTNode< DataType >::m_pLeft, and BSTNode< DataType >::m_pRight.

Referenced by BSTNode< DataType >::isLeaf().

6.2.3.3 template<typename DataType> bool BSTNode< DataType >::hasGrandparent (void) const `[virtual]`

True iff this node's parent has a parent.

Returns :
`true` if this node's parent has a parent.

Definition at line 56 of file BSTNode.cc.

References BSTNode< DataType >::hasParent(), and BSTNode< DataType >::m_p-Parent.

Referenced by BSTNode< DataType >::isZigZag(), and BSTNode< DataType >::is-ZigZig().

6.2.3.4 template<typename DataType> bool BSTNode< DataType >::hasParent (void) const [virtual]

True iff this node has a parent (i.e., if this node is *not* the root).

Returns :
`true` if this node has a parent.

Definition at line 48 of file BSTNode.cc.

References BSTNode< DataType >::m_pParent.

Referenced by BSTNode< DataType >::hasGrandparent(), BSTNode< DataType >::isLeftChild(), and BSTNode< DataType >::isRightChild().

6.2.3.5 template<typename DataType> bool BSTNode< DataType >::isLeaf (void) const [virtual]

Tests of this node is a leaf (has no childern).

Returns :
`true` iff this node is a leaf.

Definition at line 80 of file BSTNode.cc.

References BSTNode< DataType >::getNumChildren().

6.2.3.6 template<typename DataType> bool BSTNode< DataType >::isLeftChild (void) const [virtual]

Tests if this node is the left child of its parent.

If the node has no parent, it is not the left child of its parent, so this function returns false.

Returns :

`true` iff this node is a left child of its parent. `false` if this node does not have a parent.

Definition at line 64 of file BSTNode.cc.

References BSTNode< DataType >::hasParent(), BSTNode< DataType >::m_pLeft, and BSTNode< DataType >::m_pParent.

Referenced by BSTNode< DataType >::isZigZig().

6.2.3.7 template<typename DataType> bool BSTNode< DataType >::isRightChild (void) const [virtual]

Tests if this node is the right child of its parent.

If the node has no parent, it is not the right child of its parent, so this function returns false.

Returns :

`true` iff this node is a right child of its parent. `false` if this node does not have a parent.

Definition at line 72 of file BSTNode.cc.

References BSTNode< DataType >::hasParent(), BSTNode< DataType >::m_pParent, and BSTNode< DataType >::m_pRight.

Referenced by BSTNode< DataType >::isZigZig().

6.2.3.8 template<typename DataType> bool BSTNode< DataType >::isZigZag (void) const [virtual]

Test if this node has a "Zig-Zag" relation to the root of the tree.

(left child of right child, or right child of left child)

Returns :

`true` iff this node is a left child of a right child or a right child of a left child.

Definition at line 128 of file BSTNode.cc.

References BSTNode< DataType >::hasGrandparent(), and BSTNode< DataType >::isZigZig().

6.2.3.9 template<typename DataType> bool BSTNode< DataType >::isZigZig (void) const [virtual]

Test if this node has a "Zig-Zig" relation to the root of the tree.

(left child of left child, or right child of right child)

Returns :
`true` iff this node is a right child of a right child or a left child of a left child.

Definition at line 107 of file BSTNode.cc.

References BSTNode< DataType >::hasGrandparent(), BSTNode< DataType >::isLeftChild(), BSTNode< DataType >::isRightChild(), and BSTNode< DataType >::m_pParent.

Referenced by BSTNode< DataType >::isZigZag().

6.2.3.10 template<typename DataType> void BSTNode< DataType >::print (std::ostream & *out* = std::cout, std::string *filler* = "") const `[virtual]`

Pretty-Print this node and its subtree.

Parameters:
out The `ostream` to output the pretty printed tree to.

filler The filler character to use while printing. (Defaults to the emptry string. The client shouldn't mess with this.)

Definition at line 138 of file BSTNode.cc.

References BSTNode< DataType >::m_data, BSTNode< DataType >::m_pLeft, and BSTNode< DataType >::m_pRight.

6.2.3.11 template<typename DataType> BSTNode< DataType > * BSTNode< DataType >::rotateLeft (void) `[virtual]`

Rotate this node with its left child, returning a pointer to node that is now in its place.

Returns :
a pointer to the node that has been rotated to the position where the current node was.

Definition at line 207 of file BSTNode.cc.

References BSTNode< DataType >::m_pLeft, and BSTNode< DataType >::m_pRight.

6.2.3.12 template<typename DataType> BSTNode< DataType > * BSTNode< DataType >::rotateRight (void) [virtual]

Rotate this node with its right child, returning a pointer to node that is now in its place.

Returns :
a pointer to the node that has been rotated to the position where the current node was.

Definition at line 186 of file BSTNode.cc.

References BSTNode< DataType >::m_pLeft, and BSTNode< DataType >::m_pRight.

6.2.4 Member Data Documentation

6.2.4.1 template<typename DataType> DataType BSTNode< DataType >::m_data

The data contained by this node.

Definition at line 208 of file BSTNode.hh.

Referenced by BSTNode< DataType >::clone(), and BSTNode< DataType >::print().

6.2.4.2 template<typename DataType> BSTNode* BSTNode< DataType >::m_pLeft

pointer to this node's left child.

Definition at line 214 of file BSTNode.hh.

Referenced by BSTNode< DataType >::getNumChildren(), BSTNode< DataType >::isLeftChild(), BSTNode< DataType >::print(), BSTNode< DataType >::rotateLeft(), and BSTNode< DataType >::rotateRight().

6.2.4.3 template<typename DataType> BSTNode* BSTNode< DataType >::m_pParent

pointer to this node's parent.

Definition at line 211 of file BSTNode.hh.

Referenced by BinarySearchTree< KeyType, ValueType >::cloneTree(), BSTNode< DataType >::hasGrandparent(), BSTNode< DataType >::hasParent(), BSTNode<

DataType >::isLeftChild(), BSTNode< DataType >::isRightChild(), and BSTNode< DataType >::isZigZig().

6.2.4.4 template<typename DataType> BSTNode* BSTNode< DataType >::m_pRight

pointer to this node's right child.

Definition at line 217 of file BSTNode.hh.

Referenced by BSTNode< DataType >::getNumChildren(), BSTNode< DataType >::isRightChild(), BSTNode< DataType >::print(), BSTNode< DataType >::rotateLeft(), and BSTNode< DataType >::rotateRight().

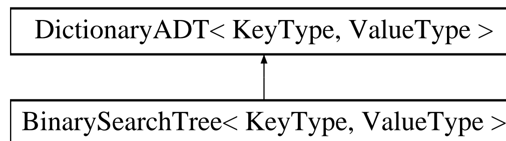
The documentation for this struct was generated from the following files:

- BSTNode.hh
- BSTNode.cc

6.3 DictionaryADT< KeyType, ValueType > Class Template Reference

```
#include <DictionaryADT.hh>
```

Inheritance diagram for DictionaryADT< KeyType, ValueType >::



6.3.1 Detailed Description

template<typename KeyType, typename ValueType> class DictionaryADT< KeyType, ValueType >

This abstract class gives the basic interface of the DictionaryADT.

A DictionaryADT is a map type structure which maps keys to values. Dupliucate keys are not allowed.

Definition at line 28 of file DictionaryADT.hh.

Public Types

- typedef std::pair< KeyType, ValueType > Entry

  Represents a Key/Value pair in the Dictionary.

- typedef std::vector< Entry > EntryList

  A list of Key/Value pairs used by the `getEntryList` *function.*

Public Methods

- virtual ~DictionaryADT ()

  Virtual destructor with trival implementation.

- virtual int getSize (void) const=0

  Returns the number of elements in the Dictionary.

- virtual bool find (const KeyType &key, ValueType &value) const=0

    Given a key, 'returns' (via a reference parameter) the value associated with it in this Dictionary.

- virtual int getEntryList (EntryList &entries) const=0

    "Returns" a list of all the Key/Value pairs in the tree.

- virtual bool insert (const KeyType &key, const ValueType &value)=0

    Inserts a new key/value pair into the Dictionary.

- virtual bool changeValue (const KeyType &oldKey, const ValueType &newValue)=0

    Changes the value associated with an existing key in the Dictionary (oldKey) to a new value.

- virtual bool insert (const Entry &entry)=0

    Same as the the other `insert` *function, but takes an* `Entry` *rather than seperate key and value arguments.*

- virtual bool changeValue (const Entry &entry)=0

    Same as the the other `ChangeValue` *function, but takes an* `Entry` *rather than seperate key and value argument.*

- virtual std::ostream & print (std::ostream &out=std::cerr) const=0

    Pretty-Print out a view of this Dictionary for debugging purposes.

6.3.2 Member Typedef Documentation

6.3.2.1 template<typename KeyType, typename ValueType> typedef std::pair<KeyType, ValueType> DictionaryADT< KeyType, ValueType >::Entry

Represents a Key/Value pair in the Dictionary.

Reimplemented in BinarySearchTree< KeyType, ValueType >.

Definition at line 36 of file DictionaryADT.hh.

6.3.2.2 template<typename KeyType, typename ValueType> typedef std::vector<Entry> DictionaryADT< KeyType, ValueType >::EntryList

A list of Key/Value pairs used by the `getEntryList` function.

Reimplemented in BinarySearchTree< KeyType, ValueType >.

Definition at line 39 of file DictionaryADT.hh.

6.3.3 Constructor & Destructor Documentation

6.3.3.1 template<typename KeyType, typename ValueType> virtual DictionaryADT< KeyType, ValueType >::~DictionaryADT () `[inline, virtual]`

Virtual destructor with trival implementation.

Why do you think this is necessary?

Definition at line 45 of file DictionaryADT.hh.

6.3.4 Member Function Documentation

6.3.4.1 template<typename KeyType, typename ValueType> virtual bool DictionaryADT< KeyType, ValueType >::changeValue (const Entry & *entry*) `[pure virtual]`

Same as the the other `ChangeValue` function, but takes an `Entry` rather than seperate key and value argument.

Returns :
`true` if the data was changed, `false` otherwise.

Parameters:
entry The entry representing the key/value information to be changed in the Dictionary.

Implemented in BinarySearchTree< KeyType, ValueType >.

6.3.4.2 template<typename KeyType, typename ValueType> virtual bool DictionaryADT< KeyType, ValueType >::changeValue (const KeyType & *oldKey*, const ValueType & *newValue*) `[pure virtual]`

Changes the value associated with an existing key in the Dictionary (oldKey) to a new value.

Assumes 'oldKey' exists in the Dictionary.

Returns :
`true` if the data was changed, `false` otherwise.

Parameters:

oldKey The key whose value will be changed.

newValue The new value to be associated with `oldKey`.

Implemented in BinarySearchTree< KeyType, ValueType >.

6.3.4.3 template<typename KeyType, typename ValueType> virtual bool DictionaryADT< KeyType, ValueType >::find (const KeyType & *key*, ValueType & *value*) const `[pure virtual]`

Given a key, 'returns' (via a reference parameter) the value associated with it in this Dictionary.

Returns :

`true` if the key is in the Dictionary, `false` otherwise. If the key is not in the Dictionary, the value of `value` is undefined.

Parameters:

key The key to search for.

value output parameter to hold a copy of the value for the given key.

Implemented in BinarySearchTree< KeyType, ValueType >.

6.3.4.4 template<typename KeyType, typename ValueType> virtual int DictionaryADT< KeyType, ValueType >::getEntryList (EntryList & *entries*) const `[pure virtual]`

"Returns" a list of all the Key/Value pairs in the tree.

The Key/Value pairs may be in any order.

Returns :

an `int` specifying the number of elements in the arrays. -1 if there was an error.

Parameters:

entries output parameter that holds the list of key/value pairs in the Dictionary. The original contents of the list is erased.

Implemented in BinarySearchTree< KeyType, ValueType >.

6.3.4.5 template<typename KeyType, typename ValueType> virtual int DictionaryADT< KeyType, ValueType >::getSize (void) const `[pure virtual]`

Returns the number of elements in the Dictionary.

Returns :
the number of elements in the dictionary.

Implemented in BinarySearchTree< KeyType, ValueType >.

6.3.4.6 template<typename KeyType, typename ValueType> virtual bool DictionaryADT< KeyType, ValueType >::insert (const Entry & *entry*) `[pure virtual]`

Same as the the other `insert` function, but takes an `Entry` rather than seperate key and value arguments.

Returns :
`true` if this key did not previously exist in the Dictionary. `false` otherwise.

Parameters:
entry The new key/value pair to be inserted into the Dictionary.

Implemented in BinarySearchTree< KeyType, ValueType >.

6.3.4.7 template<typename KeyType, typename ValueType> virtual bool DictionaryADT< KeyType, ValueType >::insert (const KeyType & *key*, const ValueType & *value*) `[pure virtual]`

Inserts a new key/value pair into the Dictionary.

If the key previously existed in the Dictionary, will overwrite the old value with the new value.

Returns :
`true` if this key did not previously exist in the tree, `false` otherwise.

Parameters:
key The new key with which `value` will be associated.

value the value to be associated with `key`.

Implemented in BinarySearchTree< KeyType, ValueType >.

6.3.4.8 template<typename KeyType, typename ValueType> virtual std::ostream& DictionaryADT< KeyType, ValueType >::print (std::ostream & *out* = std::cerr) const [pure virtual]

Pretty-Print out a view of this Dictionary for debugging purposes.

This code requires operator<< for both KeyType and ValueType, so you can remove this if you're not debugging. You can "turn off" printing by compiling in "release mode" (ie, by defining "NDEBUG")

Returns :

The ostream used by the print function. This is mainly a convienience thing.

Parameters:

out The output stream to pretty-print the Dictionary to. This funciton defaults to cerr.

Implemented in BinarySearchTree< KeyType, ValueType >.

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

- DictionaryADT.hh

Chapter 7

Project3:WordCount File Documentation

7.1 BinarySearchTree.cc File Reference

7.1.1 Detailed Description

Implementation of a templated BinarySearchTree class which maps keys to values (duplicated keys are not allowed).

Definition in file BinarySearchTree.cc.`#include "BinarySearchTree.hh"`

`#include <iostream>`

Namespaces

- namespace std

7.2 BinarySearchTree.hh File Reference

7.2.1 Detailed Description

Specification of a templated BinarySearchTree class which maps keys to values (duplicated keys are not allowed).

Definition in file BinarySearchTree.hh.#include "DictionaryADT.hh"

```
#include "BSTNode.hh"
#include <iostream>
#include <string>
```

Compounds

- class BinarySearchTree

  The BinarySearchTree class.

7.3 BSTInst.cc File Reference

7.3.1 Detailed Description

Instantiation file for the templated BST classes.

You'll want to instatiate all templated classes used by your program and any templated classes that they depend on.

Definition in file BSTInst.cc.`#include "BinarySearchTree.cc"`

```
#include "BSTNode.cc"
```

Variables

- template BinarySearchTree<int, double>

7.3.2 Variable Documentation

7.3.2.1 template BinarySearchTree<int, double>

Definition at line 16 of file BSTInst.cc.

7.4 BSTNode.cc File Reference

7.4.1 Detailed Description

Implementation of a templated BSTNode class which represents generic nodes in a Binary Search Tree.

Definition in file BSTNode.cc.#include "BinarySearchTree.hh"

```
#include <cstdlib>

#include <iostream>

#include <strstream>
```

7.5 BSTNode.hh File Reference

7.5.1 Detailed Description

Specification of a templated BSTNode containing Key/Value pairs.

Definition in file BSTNode.hh.`#include <iostream>`

`#include <string>`

Compounds

- struct BSTNode

  The Binary Search Tree node struct.

7.6 dictionary-test.cc File Reference

7.6.1 Detailed Description

Simple program for testing BST functionality (note that you can use it to test your AVL/Splay trees as well!).

Definition in file dictionary-test.cc.`#include "BinarySearchTree.hh"`

`#include <iostream>`

`#include <algorithm>`

Functions

- bool EntryValueComparator (const BinarySearchTree< int, double >::Entry &a, const BinarySearchTree< int, double >::Entry &b)

  This is a comparator which compares two entries by the ∗value∗ (the second member of the pair) rather than by the key.

- void printEntry (const BinarySearchTree< int, double >::Entry &e)

  This pretty-prints a single entry to cout.

- int main (void)

  The main test program.

7.6.2 Function Documentation

7.6.2.1 bool EntryValueComparator (const BinarySearchTree< int, double >::Entry & *a*, const BinarySearchTree< int, double >::Entry & *b*)

This is a comparator which compares two entries by the ∗value∗ (the second member of the pair) rather than by the key.

Returns :
`true` if a is less than b.

Definition at line 25 of file dictionary-test.cc.

Referenced by main().

7.6.2.2 int main (void)

The main test program.

Definition at line 50 of file dictionary-test.cc.

References BinarySearchTree< KeyType, ValueType >::changeValue(), EntryValueComparator(), BinarySearchTree< KeyType, ValueType >::find(), BinarySearchTree< KeyType, ValueType >::getEntryList(), BinarySearchTree< KeyType, ValueType >::getSize(), BinarySearchTree< KeyType, ValueType >::insert(), BinarySearchTree< KeyType, ValueType >::print(), and printEntry().

7.6.2.3 void printEntry (const BinarySearchTree< int, double >::Entry & *e*)

This pretty-prints a single entry to cout.

Yeah, so this should probably have taken some ostream reference, but I got lazy.

Parameters:

e The entry to pretty-print.

Definition at line 41 of file dictionary-test.cc.

Referenced by main().

7.7 DictionaryADT.hh File Reference

7.7.1 Detailed Description

Specification of a templated DictionaryADT class which maps keys to values (duplicated keys are not allowed).

Definition in file DictionaryADT.hh.`#include <utility>`

`#include <vector>`

`#include <iostream>`

Compounds

- class DictionaryADT

  This abstract class gives the basic interface of the DictionaryADT.

Functions

- template<typename KeyType, typename ValueType> std::ostream & operator<< (std::ostream &os, const std::pair< KeyType, ValueType > &entry)

  An overloading of operator<< that allows for easy printing of the "Entry" type in the DictionaryADT.

7.7.2 Function Documentation

7.7.2.1 template<typename KeyType, typename ValueType> std::ostream& operator<< (std::ostream & *os*, const std::pair< KeyType, ValueType > & *entry*) `[inline, static]`

An overloading of operator<< that allows for easy printing of the "Entry" type in the DictionaryADT.

We use "std::pair" rather than the typedef since the compiler seems to get confused otherwise. The function is static inline because it is in the header file so we need static to avoid multiple definition errors. It is inlined becasue it is short enough that it might as well.

Returns :

The ostream used by operator<<. This allows for chaining like: `cout << entry << endl;`.

Parameters:

os The output stream to pretty-print the entry to. This funciton defaults to cerr.

entry The entry to prettyprint.

Definition at line 173 of file DictionaryADT.hh.