CS 331 Group Project (Component A and B)

Generated by Doxygen 1.8.8

Fri Dec 12 2014 21:53:07

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 bTree< T > Class Template Reference

Public Member Functions

- bTree ()
- Node< T > * searchLeaf (Node< T > *aNodePtr, int keyToFind)
- bool treeContains (int keyToFind)
- void printRecord (int key)
- bool insertRecord (key, value)
- bool deleteRecord (int key)
- bool checkTree (bTree aTree)
- void rebuildTree (bTree &aTree)

4.1.1 Constructor & Destructor Documentation

4.1.1.1 template<class T > bTree< T >::bTree ( )

Creates a bTree

Precondition

Called onto a interior node

Postcondition

Set the child to the node which is being called by

4.1.2 Member Function Documentation

4.1.2.1 template<class T > bool bTree< T >::checkTree (bTree< T > *aTree*)

Checks the tree to see if it is balanced

Precondition

Called onto a interior node

Parameters

aTree	The tree to be checked

Postcondition

Returns true or false depeneding if the tree is balanced or not

4.1.2.2 template<class T > bool bTree< T >::deleteRecord (int *key*)

Deletes a record from the tree

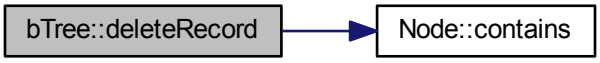
Parameters

key	the key corresponding to the record value

Postcondition

Returns true or false depending if the record was successfully deleted

Here is the call graph for this function:



4.1.2.3 template<class T > bool bTree< T >::insertRecord (key , value)

Inserts a key/record pair into the B+ Tree

Parameters

key	A key used to insert the value into the tree
value	The record to be stored into a Leaf Node

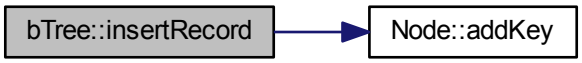
Precondition

N/A

Postcondition

Returns true or false depending if the key/value pair was successfully added

Here is the call graph for this function:



4.1.2.4 template<class T > void bTree< T >::printRecord (int *key*)

Prints a record found within the tree

Precondition

N/A

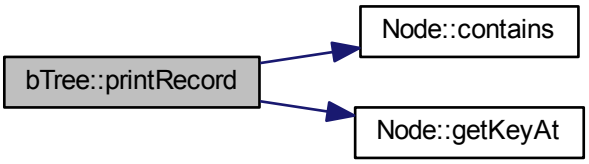
Parameters

key	The key to the record that is to be printed

Postcondition

Print all information about the record

Here is the call graph for this function:



4.1.2.5 template<class T > void bTree< T >::rebuildTree (bTree< T > & *aTree*)

Rebuild the tree to be balance

Precondition

Tree is currently unbalanced

Parameters

	post Change the tree to be balanced

4.1.2.6 template<class T > Node< T > * bTree< T >::searchLeaf (Node< T > * *aNodePtr,* int *keyToFind*)

Searches down the tree to find the Leaf Node corresponding to a key

Parameters

keyToFind	A key to find within the tree

Precondition

A treeContains() has been called before this function

Postcondition

Returns the Leaf Node containing the key

Here is the call graph for this function:

4.1.2.7 template<class T > bool bTree< T >::treeContains (int *keyToFind*)

Searches down the tree to see if a key exists within it.

Parameters

keyToFind	A key to find within the tree

Postcondition

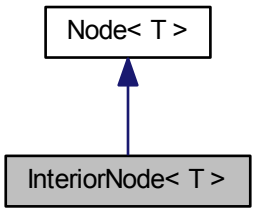
Returns true or false depending on whether the key was found or not

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

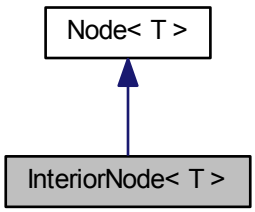
- C:/Users/Brandon/Desktop/CS 331 Final Project/bTree.h
- C:/Users/Brandon/Desktop/CS 331 Final Project/bTree.cpp

4.2 InteriorNode< T > Class Template Reference

Inheritance diagram for InteriorNode< T >:



Collaboration diagram for InteriorNode< T >:



Public Member Functions

- InteriorNode ()
- InteriorNode (int cap)
- InteriorNode (const Node< T > &newCopy)
- int getChildSize ()
- Node< T > * getChild (int key)
- void setChild (Node< T > *newChild, int position)
- void **removeChild** (int position)
- int searchKey (int key)
- int addKey (int newKey)
- void removeKey (int position)
- void split (Node< T > *&newNodePtr)
- void mergeNodes (Node< T > *&otherNodePtr)
- int **getSize** ()
- Node< T > * **getParent** ()
- void **setParent** (Node< T > *newParentPtr)
- int **getKeyAt** (int position)
- bool **contains** (int key)

Additional Inherited Members

4.2.1 Constructor & Destructor Documentation

4.2.1.1 template<class T > InteriorNode< T >::InteriorNode ( )

Default Constructor for InteriorNode

Postcondition

Creates a interiorNode

4.2.1.2 template<class T > InteriorNode< T >::InteriorNode (int *cap*)

Default Constructor for Node with capacity

Precondition

Accepts a capacity for a node, depending on what type of node is being created

Parameters

cap	capacity of the size in the node

Postcondition

Creates a base Node for either a leaf node or a interior node with a capacity

4.2.1.3 template<class T > InteriorNode< T >::InteriorNode (const Node< T > & *newCopy*)

Copy constructor that copies the node that calls it

Precondition

Accepts a node to be copied

Parameters

Node	to be copied

Postcondition

Copies the node that called the constructor

4.2.2 Member Function Documentation

4.2.2.1 template<class T > int InteriorNode< T >::addKey (int *newKey*) [virtual]

Add a key

Precondition

Node must have room for the new key. size() <= capacity

Parameters

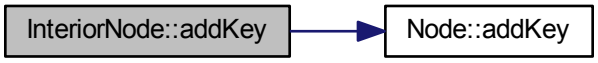
newKey	key that will be entered

Postcondition

Return true or false on if the key was successfully added

Implements Node< T >.

Here is the call graph for this function:



4.2.2.2 template<class T > Node< T > * InteriorNode< T >::getChild (int *key*) [virtual]

Returns the child where the key located

Parameters

key	key to be searched for

Postcondition

Returns the node where the key is located

Implements Node< T >.

4.2.2.3 template<class T > int InteriorNode< T >::getChildSize ( ) [virtual]

Get size of the node

Postcondition

Returns the size of the node

Reimplemented from Node< T >.

4.2.2.4 template<class T > void InteriorNode< T >::mergeNodes (Node< T > *& *otherNodePtr*) [virtual]

Merge two nodes together

Parameters

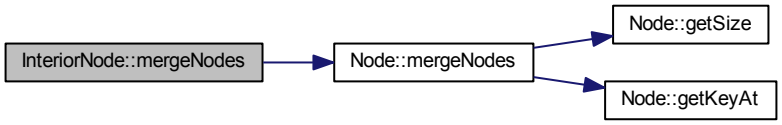
otherNodePtr	Pointer of the node the be merged with

Postcondition

Merges the two nodes together, placing all keys into the orinigal node

Implements Node< T >.

Here is the call graph for this function:



4.2.2.5 template<class T > void InteriorNode< T >::removeKey (int *position*)

Remove a key

Precondition

Key must be located in the node

Parameters

position	Location of the key in the vector

Postcondition

Removes the key from the vector

4.2.2.6 template<class T > int InteriorNode< T >::searchKey (int *key*) [virtual]

Searches for a key in a node

Precondition

Called from a interior node

Parameters

key	The key to be searched for

Postcondition

Returns the position of the key

Reimplemented from Node< T >.

4.2.2.7 template<class T > void InteriorNode< T >::setChild (Node< T > * *newChild,* int *position*)

Sets the child of a node

Precondition

Called onto a interior node

Parameters

newChild	The child to be set
position	Position of the node

Postcondition

Set the child to the node which is being called by

4.2.2.8 template<class T > void InteriorNode< T >::split (Node< T > *& *newNodePtr*) [virtual]

Split node

Parameters

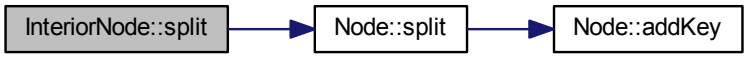
newNodePtr	Pointer to the new node that is being created

Postcondition

Splits the node by placing half of the key into a new node

Implements Node< T >.

Here is the call graph for this function:

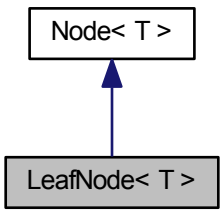


The documentation for this class was generated from the following files:

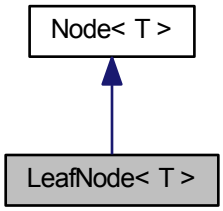
- C:/Users/Brandon/Desktop/CS 331 Final Project/InteriorNode.h
- C:/Users/Brandon/Desktop/CS 331 Final Project/InteriorNode.cpp

4.3 LeafNode< T > Class Template Reference

Inheritance diagram for LeafNode< T >:



Collaboration diagram for LeafNode< T >:



Public Member Functions

- LeafNode ()
- LeafNode (int cap)
- LeafNode (const Node< T > &newCopy)
- int addKey (int newKey)
- Node< T > * getChild (int key)
- void setChild (Node< T > *newChild, int position)
- void split (Node< T > *&newNodePtr)
- void mergeNodes (Node< T > *&otherNodePtr)
- int **getSize** ()
- Node< T > * **getParent** ()
- void **setParent** (Node< T > *newParentPtr)
- int **getKeyAt** (int position)
- bool **contains** (int key)

Additional Inherited Members

4.3.1 Constructor & Destructor Documentation

4.3.1.1 template<class T > LeafNode< T >::LeafNode ( )

Default Constructor for LeafNode

Postcondition

Creates a leaf node

4.3.1.2 template<class T > LeafNode< T >::LeafNode (int *cap*)

Default Constructor for LeafNode

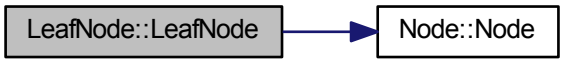
Parameters

cap	Capicity of the node

Postcondition

Creates a leaf node with a set capicity

Here is the call graph for this function:



4.3.1.3 template<class T > LeafNode< T >::LeafNode (const Node< T > & *newCopy*)

Copy constructor that copies the node that calls it

Precondition

Accepts a node to be copied

Parameters

Node	to be copied

Postcondition

Copies the node that called the constructor

4.3.2 Member Function Documentation

4.3.2.1 template<class T > int LeafNode< T >::addKey (int *newKey*) [virtual]

Add a key

Precondition

Node must have room for the new key. size() <= capacity

Parameters

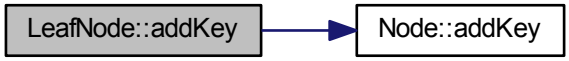
newKey	key that will be entered

Postcondition

Return true or false on if the key was successfully added into the vector of keys

Implements Node< T >.

Here is the call graph for this function:



4.3.2.2 template<class T > Node< T > * LeafNode< T >::getChild (int *key*) [virtual]

Returns the child where the key located

Parameters

key	key to be searched for

Postcondition

Returns the node where the key is located

Implements Node< T >.

4.3.2.3 template<class T > void LeafNode< T >::mergeNodes (Node< T > *& *otherNodePtr*) [virtual]

Merge two nodes together

Parameters

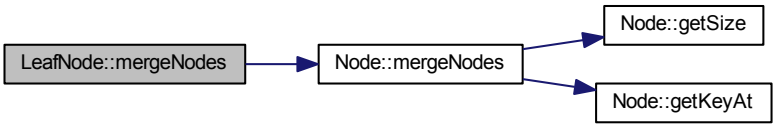
otherNodePtr	Pointer of the node the be merged with

Postcondition

Merges the two nodes together, placing all keys into the orinigal node

Implements Node< T >.

Here is the call graph for this function:



4.3.2.4 template<class T > void LeafNode< T >::setChild (Node< T > * *newChild,* int *position*)

Sets the child of a node

Precondition

Called onto a interior node

Parameters

newChild	The child to be set
position	Position of the node

Postcondition

Set the child to the node which is being called by

4.3.2.5 template<class T > void LeafNode< T >::split (Node< T > *& *newNodePtr*) [virtual]

Split node

Parameters

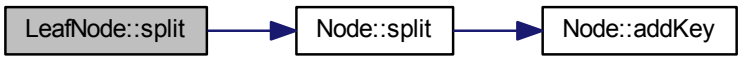
newNodePtr	Pointer to the new node that is being created

Postcondition

Splits the node by placing half of the key into a new node

Implements Node< T >.

Here is the call graph for this function:

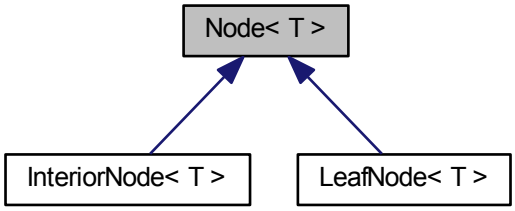


The documentation for this class was generated from the following files:

- C:/Users/Brandon/Desktop/CS 331 Final Project/LeafNode.h
- C:/Users/Brandon/Desktop/CS 331 Final Project/LeafNode.cpp

4.4 Node< T > Class Template Reference

Inheritance diagram for Node< T >:



Public Member Functions

- Node ()
- Node (int cap)
- Node (const Node &newCopy)
- int getKeyAt (int position)
- virtual int searchKey (int key)
- virtual int addKey (int newKey)=0
- Node< T > * getParent ()
- void setParent (Node< T > *newParentPtr)
- int getSize ()
- virtual int getChildSize ()
- virtual Node< T > * **getChild** (int key)=0
- bool contains (int key)
- virtual void split (Node< T > *&newNodePtr)=0
- virtual void mergeNodes (Node< T > *&otherNodePtr)=0

Protected Attributes

- vector< int > **keys**
- Node< T > * **parentPtr**
- int **capacity**

4.4.1 Constructor & Destructor Documentation

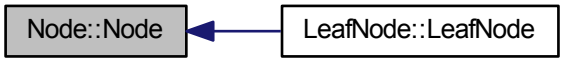
4.4.1.1 template<class T > Node< T >::Node ( )

Default Constructor for Node

Postcondition

Creates a base Node for either a leaf node or a interior node

Here is the caller graph for this function:



4.4.1.2 template<class T > Node< T >::Node (int *cap*)

Default Constructor for Node with capacity

Precondition

Accepts a capacity for a node, depending on what type of node is being created

Parameters

cap	capacity of the size in the node

Postcondition

Creates a base Node for either a leaf node or a interior node with a capacity

4.4.1.3 template<class T > Node< T >::Node (const Node< T > & *newCopy*)

Copy constructor that copies the node that calls it

Precondition

Accepts a node to be copied

Parameters

Node	to be copied

Postcondition

Copies the node that called the constructor

4.4.2 Member Function Documentation

4.4.2.1 template<class T > int Node< T >::addKey (int *newKey*) [pure virtual]

Add a key

Precondition

Node must have room for the new key. size() <= capacity

Parameters

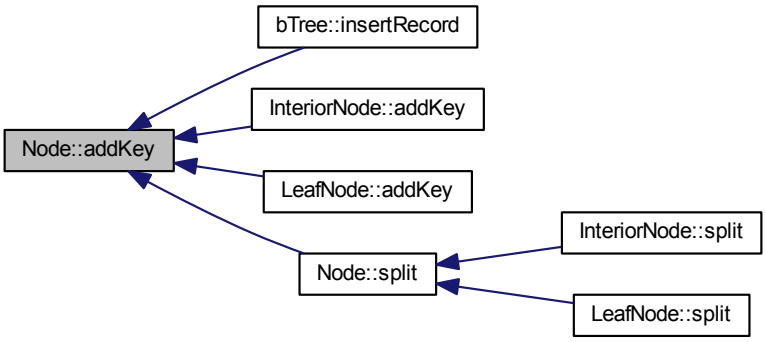
newKey	key that will be entered

Postcondition

Return true or false on if the key was successfully added

Implemented in InteriorNode< T >, and LeafNode< T >.

Here is the caller graph for this function:



4.4.2.2 template<class T > bool Node< T >::contains (int *key*)

Checks to see if a key is contained in the node

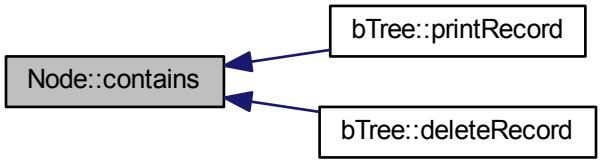
Parameters

key	key that will be searched for

Postcondition

Return true or false depending if the key was found

Here is the caller graph for this function:



4.4.2.3 template<class T > int Node< T >::getChildSize ( ) [virtual]

Get the size of the nodes child

Postcondition

Return the size the child node

Reimplemented in InteriorNode< T >.

Here is the caller graph for this function:

4.4.2.4 template<class T > int Node< T >::getKeyAt (int *position*)

Returns the key at its position

Precondition

Position must be less than keys.size()

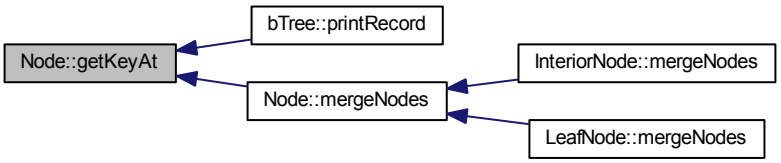
Parameters

position	position of key that is being called

Postcondition

Returns the position of the key being searched

Here is the caller graph for this function:



4.4.2.5 template<class T > Node< T > * Node< T >::getParent ( )

Get Parent of Node

Postcondition

Return the parent of the node, otherwise will return NULL

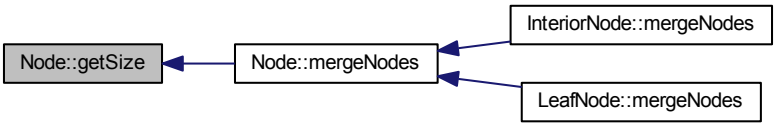
4.4.2.6 template<class T > int Node< T >::getSize ( )

Get the size of the node

Postcondition

Return the number of items in the node

Here is the caller graph for this function:



4.4.2.7 template<class T > void Node< T >::mergeNodes (Node< T > *& *otherNodePtr*) [pure virtual]

Merge two nodes together

Parameters

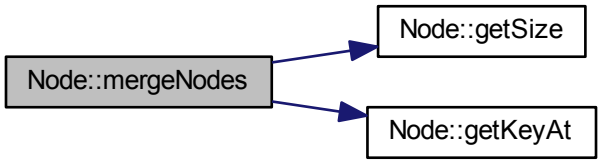
otherNodePtr	Pointer of the node the be merged with

Postcondition

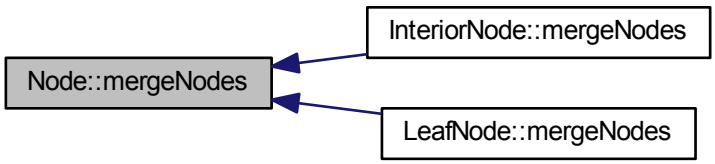
Merges the two nodes together, placing all keys into the orinigal node

Implemented in InteriorNode< T >, and LeafNode< T >.

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.2.8 template<class T > int Node< T >::searchKey (int *key*) [virtual]

Search Keys

Parameters

key	key that will be searched for

Postcondition

Return the position of the key in the vector of keys

Reimplemented in InteriorNode< T >.

4.4.2.9 template<class T > void Node< T >::setParent (Node< T > * *newParentPtr*)

Set Parent to a Node

Parameters

newParentPtr	pointer to be set as parent to a node

Postcondition

Sets the parent to the node to newParentPtr

4.4.2.10 template<class T > void Node< T >::split (Node< T > *& *newNodePtr*) [pure virtual]

Split Node

Parameters

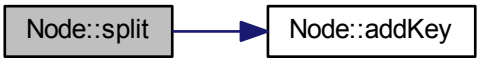
newNodePtr	pointer to node that will be split, provided by the B+ Tree class

Postcondition

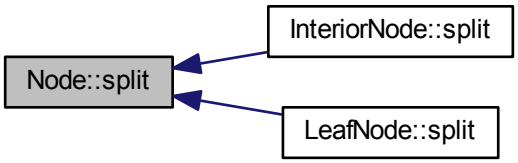
Creates a new node and places half the keys into the new node

Implemented in InteriorNode< T >, and LeafNode< T >.

Here is the call graph for this function:



Here is the caller graph for this function:



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

- C:/Users/Brandon/Desktop/CS 331 Final Project/Node.h
- C:/Users/Brandon/Desktop/CS 331 Final Project/Node.cpp

Chapter 5

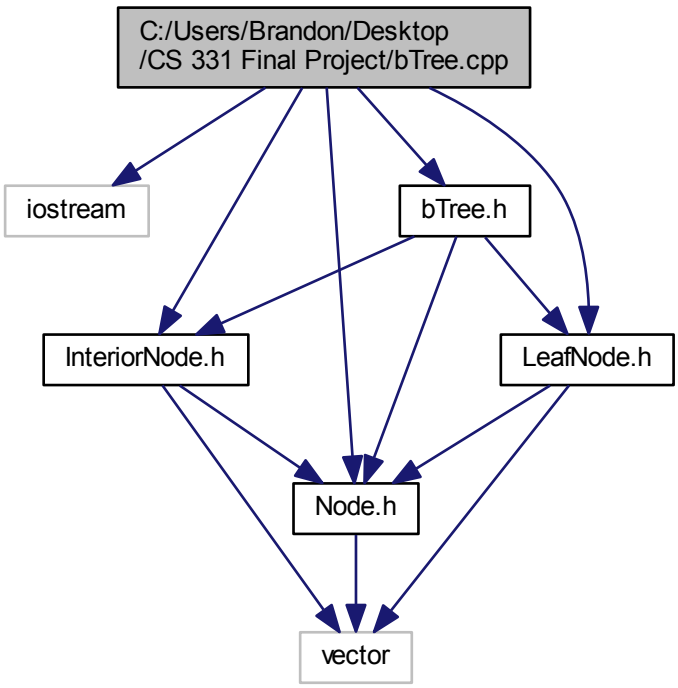
File Documentation

5.1 C:/Users/Brandon/Desktop/CS 331 Final Project/bTree.cpp File Reference

Implementation of bTree class.

```
#include <iostream>
#include "Node.h"
#include "InteriorNode.h"
#include "LeafNode.h"
#include "bTree.h"
```

Include dependency graph for bTree.cpp:



5.1.1 Detailed Description

Implementation of bTree class.

Author

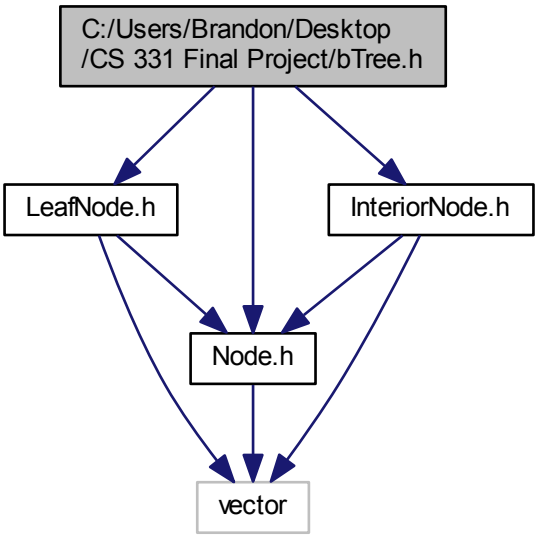
Brandon Theisen, Jason Pederson, Kelvin Schutz, Chris Scholl, Jared Kareniemi

5.2 C:/Users/Brandon/Desktop/CS 331 Final Project/bTree.h File Reference

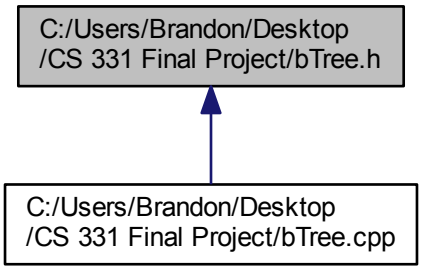
bTree class

```
#include "Node.h"
#include "LeafNode.h"
#include "InteriorNode.h"
```

Include dependency graph for bTree.h:



This graph shows which files directly or indirectly include this file:



Classes

- class bTree< T >

5.2.1 Detailed Description

bTree class

Author

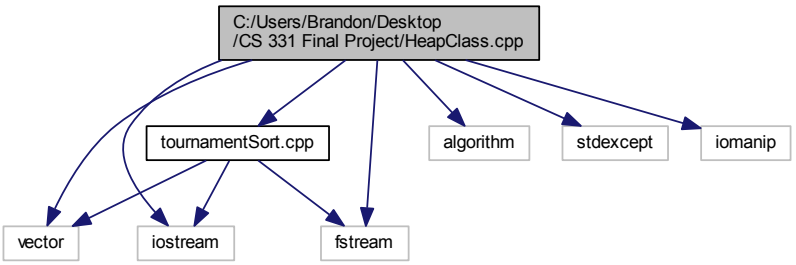
Brandon Theisen, Jason Pederson, Kelvin Schutz, Chris Scholl, Jared Kareniemi

5.3 C:/Users/Brandon/Desktop/CS 331 Final Project/HeapClass.cpp File Reference

Replacment Selection Sort.

```
#include <vector>
#include <iostream>
#include <algorithm>
#include <stdexcept>
#include "tournamentSort.cpp"
#include <fstream>
#include <iomanip>
```

Include dependency graph for HeapClass.cpp:



Functions

- template<class T >
  void activeHeapUp (vector< T > &vect)
- template<class T >
  void activeHeapDown (vector< T > &vect)
- template<class T >
  void pendingHeapUp (vector< T > &vect)
- void **emptyActiveHeap** ()
- template<class T >
  void printVector (vector< T > &vec)
- template<class T >
  void printVectorList (vector< vector< T > > &vectList)
- template<class T >
  void emptyActiveHeap (vector< T > &outVect, vector< T > &contVect, vector< vector< T > > &outVectList)
- template<class T >
  void **replacementSort** (vector< T > &inputVector, string outputName)

Variables

- int **active_heap_size** = 0
- bool **working** = true

5.3.1 Detailed Description

Replacment Selection Sort.

Author

Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

5.3.2 Function Documentation

5.3.2.1 template<class T > void activeHeapDown (vector< T > & *vect*)

Heapify the active heap

Precondition

Accepts a vector containing integers or strings

Parameters

vect	vector containing integers or strings

Postcondition

Sorts the element downwards

Here is the caller graph for this function:

5.3.2.2 template<class T > void activeHeapUp (vector< T > & *vect*)

Heapify the active heap so insert smallest elment on the top

Precondition

Accepts a vector containing integers or strings

Parameters

vect	vector containing integers or strings

Postcondition

Inserts the smallest element to the top of the active heap

5.3.2.3 template<class T > void emptyActiveHeap (vector< T > & *outVect,* vector< T > & *contVect,* vector< vector< T > > & *outVectList*)

Heapify the active heap so insert smallest elment on the top

Precondition

Active heap contaings zero elements

Here is the call graph for this function:

5.3.2.4 template<class T > void pendingHeapUp (vector< T > & *vect*)

Heapify the pending heap

Precondition

Accepts a vector containing integers or strings

Parameters

vect	vector containing integers or strings

Postcondition

Inserts last element to the top of the pending heap

5.3.2.5 template<class T > void printVector (vector< T > & *vec*)

Prints the vector to output screen

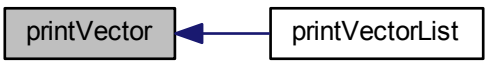
Precondition

Accepts a vector containing integers or strings

Parameters

vect	vector containing integers or strings

Here is the caller graph for this function:



5.3.2.6 template<class T > void printVectorList (vector< vector< T > > & *vectList*)

Prints the list of vectors to the output screen

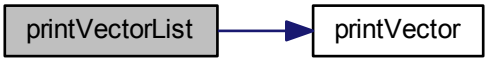
Precondition

Accepts a list of vectors containing integers or strings

Parameters

vect	vector containing integers or strings

Here is the call graph for this function:

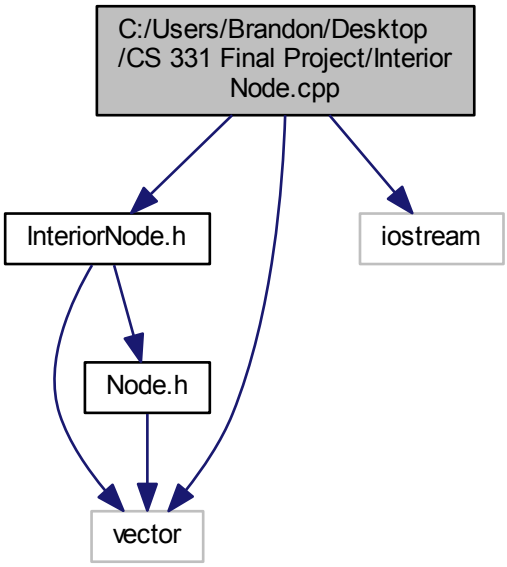


5.4 C:/Users/Brandon/Desktop/CS 331 Final Project/InteriorNode.cpp File Reference

Implementation of InteriorNode class.

```
#include "InteriorNode.h"
#include <vector>
#include <iostream>
```

Include dependency graph for InteriorNode.cpp:



5.4.1 Detailed Description

Implementation of InteriorNode class.

Author

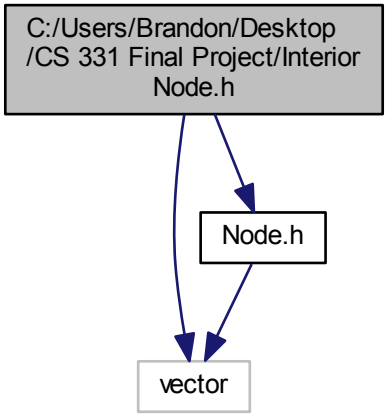
Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

5.5 C:/Users/Brandon/Desktop/CS 331 Final Project/InteriorNode.h File Reference

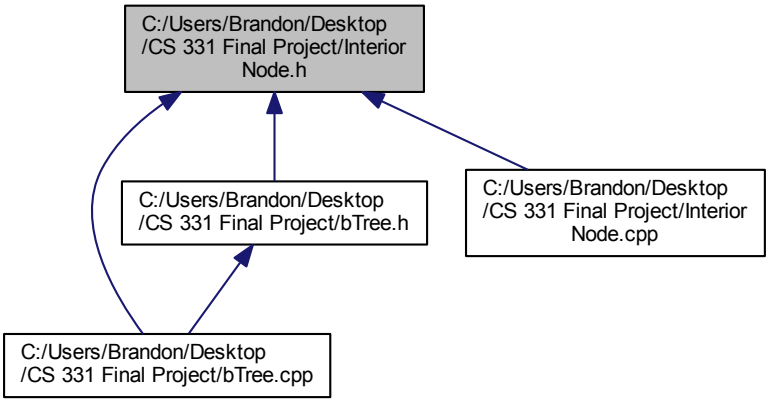
InteriorNode class.

```
#include <vector>
#include "Node.h"
```

Include dependency graph for InteriorNode.h:



This graph shows which files directly or indirectly include this file:



Classes

- class InteriorNode< T >

5.5.1 Detailed Description

InteriorNode class.

Author

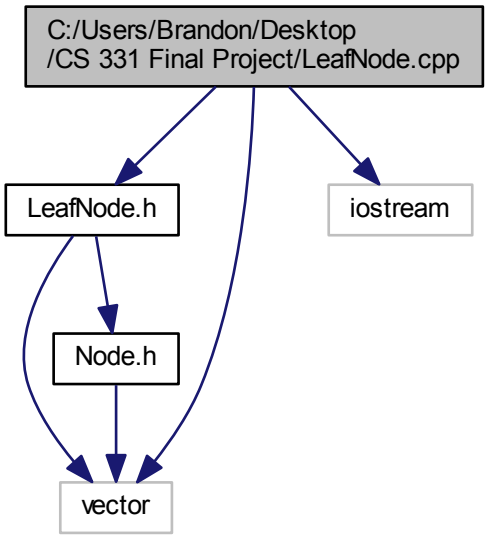
Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

5.6 C:/Users/Brandon/Desktop/CS 331 Final Project/LeafNode.cpp File Reference

Implementation of LeafNode class.

```
#include "LeafNode.h"
#include <vector>
#include <iostream>
```

Include dependency graph for LeafNode.cpp:



5.6.1 Detailed Description

Implementation of LeafNode class.

Author

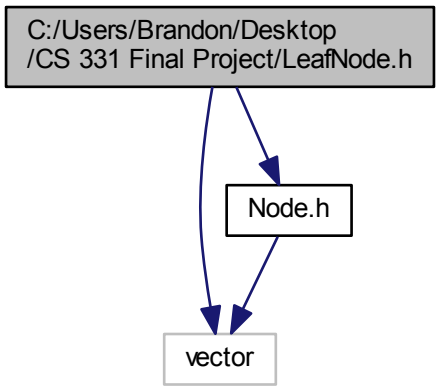
Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

5.7 C:/Users/Brandon/Desktop/CS 331 Final Project/LeafNode.h File Reference

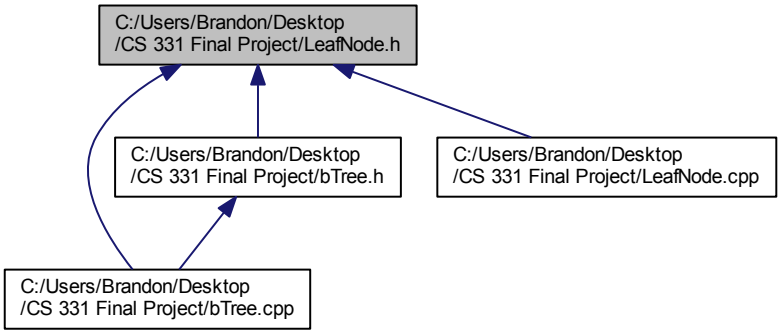
LeafNode class.

```
#include <vector>
#include "Node.h"
```

Include dependency graph for LeafNode.h:



This graph shows which files directly or indirectly include this file:



Classes

- class LeafNode< T >

5.7.1 Detailed Description

LeafNode class.

Author

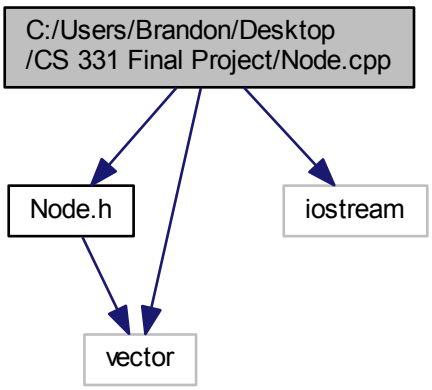
Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

5.8 C:/Users/Brandon/Desktop/CS 331 Final Project/Node.cpp File Reference

Implementation of base Node Class.

```
#include "Node.h"
#include <vector>
#include <iostream>
```

Include dependency graph for Node.cpp:



5.8.1 Detailed Description

Implementation of base Node Class.

Author

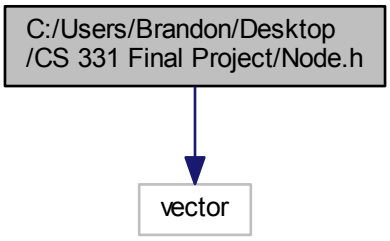
Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

5.9 C:/Users/Brandon/Desktop/CS 331 Final Project/Node.h File Reference

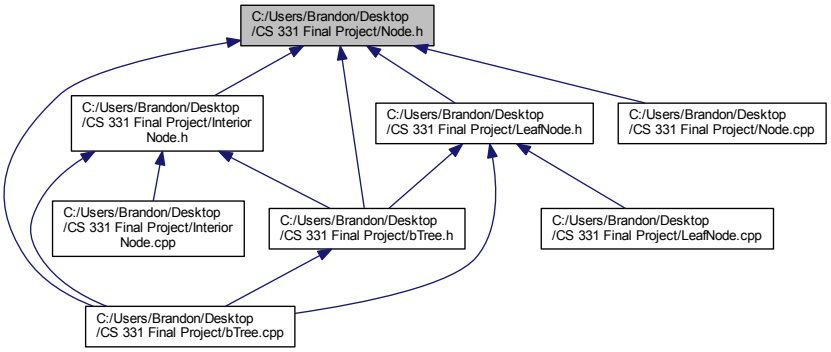
Base Node Class.

```
#include <vector>
```

Include dependency graph for Node.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Node< T >

5.9.1 Detailed Description

Base Node Class.

Author

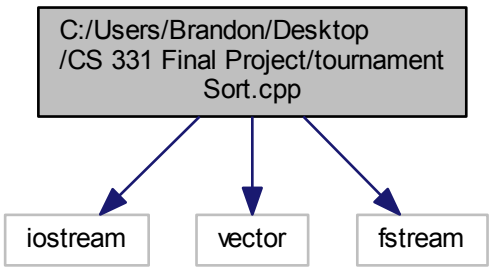
Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

5.10 C:/Users/Brandon/Desktop/CS 331 Final Project/tournamentSort.cpp File Reference

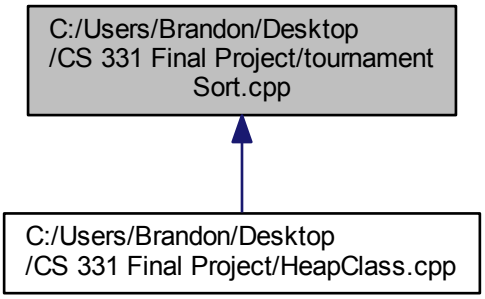
Tournament Sort and Test Program.

```
#include <iostream>
#include <vector>
#include <fstream>
```

Include dependency graph for tournamentSort.cpp:



This graph shows which files directly or indirectly include this file:



Functions

- template<class T >
  vector< T > tournamentSort (vector< vector< T > > &v)
- template<class T >
  void **printToFile** (vector< T > v, ostream &outputFile, string outputName)
- template<class T >
  void printToFile (vector< T > v, ofstream &outputFile, string outputName)

5.10.1 Detailed Description

Tournament Sort and Test Program.

Author

Brandon Theisen, Jason Pederson, Kelvin Shultz, Chris, Jared

5.10.2 Function Documentation

5.10.2.1 template<class T > void printToFile (vector< T > *v,* ofstream & *outputFile,* string *outputName*)

Outputs sorted list to file

Precondition

Accepts a a vector of a sorted complete list

Parameters

v	sorted vector of a complete list of vectors or integers
outputFile	output file to be written to

Postcondition

return is void

5.10.2.2 template<class T > vector< T > tournamentSort (vector< vector< T > > & *v*)

Uses tournament sort to sort a list of vectors

Precondition

Accepts a list of vectors that contains sorted integers or strings

Parameters

v	vector of vectors that contain sorted integers or strings

Postcondition

returns a vector of sorted strings or integers

Index