BinaryTrees1

0.2.0

Generated by Doxygen 1.8.17

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

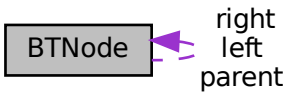
Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 BTNode Class Reference

Collaboration diagram for BTNode:



Public Member Functions

- BTNode (int dataVal)
- char nodeName ()
- int nodeData ()
- BTNode ()
- int nodeNum ()

Public Attributes

- BTNode * left
- BTNode * right
- BTNode * parent
- int num

Static Public Attributes

- static int count = 0

3.1.1 Detailed Description

Binary Tree Node

This is from Open Data Structures in C++ by Pat Morin

Definition at line 21 of file binSearch.cpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BTNode() [1/2]

```
BTNode::BTNode (
            int dataVal )  [inline]
```

BTNode constructor

Definition at line 30 of file binSearch.cpp.

```
30                          {
31          cout « "name = " « name « endl;
32          left = NULL;
33          right = NULL;
34          parent = NULL;
35          objName = name++;
36          data = dataVal;
37      }
```

3.1.2.2 BTNode() [2/2]

```
BTNode::BTNode ( )  [inline]
```

BTNode constructor

Definition at line 29 of file main.cpp.

```
29              {
30        left = NULL;
31        right = NULL;
32        parent = NULL;
33        num = count++;
34    }
```

3.1.3 Member Function Documentation

3.1.3.1 nodeData()

```
int BTNode::nodeData ( )  [inline]
```

This reports the node's data

Definition at line 49 of file binSearch.cpp.

```
49                      {
50          return(data);
51      }
```

3.1.3.2 nodeName()

```
char BTNode::nodeName ( )  [inline]
```

This reports the node's name

Definition at line 42 of file binSearch.cpp.

```
42                      {
43          return(objName);
44      }
```

3.1.3.3 nodeNum()

```
int BTNode::nodeNum ( )  [inline]
```

This reports the node's number

Definition at line 39 of file main.cpp.

```
39                      {
40          return(num);
41      }
```

3.1.4 Member Data Documentation

3.1.4.1 count

```
int BTNode::count = 0  [static]
```

Definition at line 24 of file main.cpp.

3.1.4.2 left

```
BTNode * BTNode::left
```

Definition at line 23 of file binSearch.cpp.

3.1.4.3 num

```
int BTNode::num
```

Definition at line 23 of file main.cpp.

3.1.4.4 parent

```
BTNode * BTNode::parent
```

Definition at line 25 of file binSearch.cpp.

3.1.4.5 right

```
BTNode * BTNode::right
```

Definition at line 24 of file binSearch.cpp.

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

- /home/addis/Trees2/src/binSearch.cpp
- /home/addis/Trees2/src/main.cpp

Chapter 4

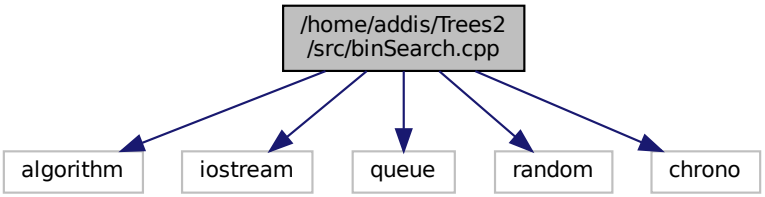
File Documentation

4.1 /home/addis/Trees2/src/binSearch.cpp File Reference

This is a demonstration of binary search trees.

```
#include <algorithm>
#include <iostream>
#include <queue>
#include <random>
#include <chrono>
```

Include dependency graph for binSearch.cpp:



Classes

- class BTNode

Functions

- BTNode * addNode (BTNode *rootNode, BTNode *n)
- BTNode * addNode (BTNode *rootNode, int dataval)
- BTNode * genExampleTree (BTNode *root)
- BTNode * genTree (vector< int > value)
- int height (BTNode *u)
- int minHeight (BTNode *u)
- void randTreeTest (int M, int N)
- void printTree (BTNode *rootNode)
- void printBT (const string &prefix, BTNode *node, bool isLeft)
- void printBT (BTNode *node)
- int main (int, char **)

4.1.1 Detailed Description

This is a demonstration of binary search trees.

This is a demo from CPTR 227 class

Author

Addis Bogale

Date

2021 March 23

4.1.2 Function Documentation

4.1.2.1 addNode() [1/2]

```
BTNode* addNode (
            BTNode * rootNode,
            BTNode * n )
```

This function adds a node to a binary search tree.

Parameters

rootNode	is the pointer to the tree's root node
n	is the node to add

Returns

pointer to rootNode if successful, NULL otherwise

Definition at line 69 of file binSearch.cpp.

```
69                                              {
70     BTNode* prev = NULL;
71     BTNode* w = rootNode;
72     if (rootNode == NULL) { // starting an empty tree
73         rootNode = n;
74     }
75     else {
76         // Find the node n belongs under, prev, n's new parent
77         while (w != NULL) {
78             prev = w;
79             if (n->nodeData() < w->nodeData()) {
80                 w = w->left;
81             }
82             else if (n->nodeData() > w->nodeData()) {
83                 w = w->right;
84             }
85             else { // data already in the tree
86                 return(NULL);
87             }
88         }
89         // now prev should contain the node that should be n's parent
90         // Add n to prev
```

```
91          if (n->nodeData() < prev->nodeData()) {
92              prev->left = n;
93          }
94          else {
95              prev->right = n;
96          }
97      }
98      return(rootNode);
99  }
```

4.1.2.2 addNode() [2/2]

```
BTNode* addNode (
              BTNode * rootNode,
              int dataval )
```

Adds a new node with the passed data value

Parameters

rootNode	pointer to root node
dataval	an integer for the new node's data

Returns

pointer to root node or NULL if not successful

Definition at line 109 of file binSearch.cpp.

```
109                                                 {
110     BTNode* newNode = new BTNode(dataval);
111     if (addNode(rootNode, newNode) == NULL) {
112         cout « dataval « " already in tree" « endl;
113     }
114     else {
115         cout « dataval « " succesfully added" « endl;
116     }
117     return(rootNode);
118 }
```

4.1.2.3 genExampleTree()

```
BTNode* genExampleTree (
              BTNode * root )
```

This generates a simple tree to play with

It is a bit of a hack.

Definition at line 125 of file binSearch.cpp.

```
125                                             {
126     //int inData[] = {1,2,3,4,5,6,7};
127     int inData[] = { 4,6,5,7,2,1,3 };
128     int classData[] = { 1,3,4,5,6,7,8,9,11,12,13,14 };
129     for (int ii = 0; ii < 7; ii++) {
130         addNode(root, inData[ii]);
131     }
132     return root;
133 }
```

4.1.2.4 genTree()

```
BTNode* genTree (
            vector< int > value )
```

Definition at line 136 of file binSearch.cpp.

```
136                                         {
137     BTNode* head = new BTNode(value[0]);
138     for (int ii = 1; ii < value.size(); ii++) {
139         addNode(head, value[ii]);
140     }
141
142     return head;
143 }
```

4.1.2.5 height()

```
int height (
            BTNode * u )
```

Definition at line 146 of file binSearch.cpp.

```
146                         {
147     if (u == NULL) {
148         cout « "Reached NULL end of branch" « endl;
149         return(-1);
150     }
151     //cout « "Calculating the height of node " « u->nodeNum() « endl;
152     return(1 + max(height(u->left), height(u->right)));
153     //cout « (1 + max(height(u->left), height(u->right)));
154 }
```

4.1.2.6 main()

```
int main (
            int ,
            char **  )
```

Definition at line 254 of file binSearch.cpp.

```
254                         {
255     BTNode* rootNode = new BTNode(0); // pointer to the root node
256     genExampleTree(rootNode);
257     printBT(rootNode);
258     printTree(rootNode);
259     randTreeTest(100, 26);
260     // height(rootNode);
261
262 }
```

4.1.2.7 minHeight()

```
int minHeight (
            BTNode * u )
```

Definition at line 157 of file binSearch.cpp.

```
157                           {
158     if (u == NULL) {
159         cout « "Reached NULL end of branch" « endl;
160         return(-1);
161     }
162     //cout « "Calculating the height of node " « u->nodeNum() « endl;
163     return(1 + min(height(u->left), height(u->right)));
164     //cout « (1 + max(height(u->left), height(u->right)));
165
166 }
```

4.1.2.8 printBT() [1/2]

```
void printBT (
            BTNode * node )
```

An overload to simplify calling printBT

Parameters

node	is the root node of the tree to be printed

Definition at line 248 of file binSearch.cpp.

```
249 {
250     printBT("", node, false);
251 }
```

4.1.2.9 printBT() [2/2]

```
void printBT (
            const string & prefix,
            BTNode * node,
            bool isLeft )
```

Print a binary tree

This example is modified from: https://stackoverflow.com/a/51730733

Parameters

prefix	is a string of characters to start the line with
node	is the current node being printed
isLeft	bool true if the node is a left node

Definition at line 225 of file binSearch.cpp.

```
226 {
227     if (node != NULL)
228     {
229         cout « prefix;
230 
231         cout « (isLeft ? "|--" : "--");
232 
233         // print the value of the node
234         //cout « node->nodeName() « ':' « node->nodeData() « std::endl;
235         cout « node->nodeData() « std::endl;
236 
237         // enter the next tree level - left and right branch
238         printBT(prefix + (isLeft ? "|   " : "    "), node->left, true);
239         printBT(prefix + (isLeft ? "|   " : "    "), node->right, false);
240     }
241 }
```

4.1.2.10 printTree()

```
void printTree (
            BTNode * rootNode )
```

Prints out a representtation of a binary search tree

Parameters

rootNode	is a pointer to the root node

Definition at line 190 of file binSearch.cpp.

```
190                                 {
191     queue<BTNode*> todo; // the queue of nodes left to visit
192     BTNode* cur; // current node
193     BTNode* prev; // The previous node
194 
195     todo.push(rootNode);
196 
197     while (!todo.empty()) {
198         cur = todo.front();
199         // Print current node
200         cout « cur->nodeName() « ':' « cur->nodeData() « '\t';
201         // add cur->left to queue
202         if (cur->left != NULL) {
203             todo.push(cur->left);
204         }
205         // add cur->right to queue
206         if (cur->right != NULL) {
207             todo.push(cur->right);
208         }
209         // remove cur from queue
210         todo.pop();
211     }
212     cout « endl;
213 }
```

4.1.2.11 randTreeTest()

```
void randTreeTest (
              int M,
              int N )
```

Definition at line 167 of file binSearch.cpp.

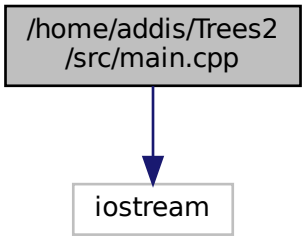
```
167                                 {
168     unsigned seed = chrono::system_clock::now().time_since_epoch().count();
169     for (int ii = 0; ii < M; ii++) {
170         BTNode* head = new BTNode(5);
171         vector <int> value;
172         for (int i = 0; i < N; i++) {
173             value.push_back(i);
174         }
175         default_random_engine generator(seed);
176         shuffle(value.begin(), value.end(), generator);
177         height(head);
178         minHeight(head);
179     }
180 
181 
182 }
```

4.2 /home/addis/Trees2/src/main.cpp File Reference

This is a demonstration of simple binary trees.

```
#include <iostream>
```

Include dependency graph for main.cpp:



Classes

- class BTNode

Functions

- int depth (BTNode ∗u)
- void traverse (BTNode ∗rootNode)
- void nonRecursiveTraverse (BTNode ∗rootNode)
- int height (BTNode ∗u)
- BTNode ∗ genExampleTree (BTNode ∗root)
- int main (int, char ∗∗)

4.2.1 Detailed Description

This is a demonstration of simple binary trees.

This is a demo from CPTR 227 class

Author

Seth McNeill

Date

2021 February 24

4.2.2 Function Documentation

4.2.2.1 depth()

```
int depth (
            BTNode * u )
```

Calculates the depth (number of steps between node and root) of a node

Parameters

pointer	to BTNode to measure the depth of

Returns

integer count of depth

Definition at line 54 of file main.cpp.

```
54                          {
55     int d = 0; // depth counter
56     while(u != NULL) {
57         u = u->parent;
58         d++;
59     }
60     return(--d);
61 }
```

4.2.2.2 genExampleTree()

```
BTNode* genExampleTree (
            BTNode * root )
```

This generates a simple tree to play with

It is a bit of a hack.

Definition at line 134 of file main.cpp.

```
134                                         {
135     BTNode* one = new BTNode();
136     BTNode* two = new BTNode();
137     BTNode* three = new BTNode();
138     BTNode* four = new BTNode();
139     BTNode* five = new BTNode();
140     BTNode* six = new BTNode();
141     cout « "Created the nodes" « endl;
142     root->left = one;
143     cout « "Added root->left" « endl;
144     one->parent = root;
145     root->right = two;
146     two->parent = root;
147     two->left = three;
148     three->parent = two;
149     two->right = four;
150     four->parent = two;
151     one->left = five;
152     five->parent = one;
153     five->left = six;
154     six->parent = five;
155     cout « "root's number: " « root->nodeNum() « endl;
156     cout « "one's number: " « one->nodeNum() « endl;
157     cout « "two's number: " « two->nodeNum() « endl;
158     cout « "three's number: " « three->nodeNum() « endl;
159     cout « "four's number: " « four->nodeNum() « endl;
160     cout « "five's number: " « five->nodeNum() « endl;
161     cout « "six's number: " « six->nodeNum() « endl;
162     cout « "six's depth is " « depth(six) « endl;
163     cout « "root's height is " « height(root) « endl;
164     return root;
165 }
```

4.2.2.3 height()

```
int height (
            BTNode * u )
```

This calculates the height (max number of steps until leaf node)

Parameters

pointer	to a BTNode

Returns

integer count of height

Definition at line 120 of file main.cpp.

```
120                         {
121     if (u == NULL) {
122         cout « "Reached NULL end of branch" « endl;
123         return(-1);
124     }
125     cout « "Calculating the height of node " « u->nodeNum() « endl;
126     return(1 + max(height(u->left), height(u->right)));
127 }
```

4.2.2.4 main()

```
int main (
              int ,
              char **  )
```

Definition at line 168 of file main.cpp.

```
168                         {
169     BTNode* rootNode = new BTNode(); // pointer to the root node
170     genExampleTree(rootNode);
171     cout « endl « "Traversing the binary tree" « endl;
172     traverse(rootNode);
173     cout « endl « "Non-recursive traversing" « endl;
174     nonRecursiveTraverse(rootNode);
175 }
```

4.2.2.5 nonRecursiveTraverse()

```
void nonRecursiveTraverse (
              BTNode * rootNode )
```

Traverses all nodes in a binary tree non-recursively

Parameters

A	pointer to the root node of interest

Definition at line 85 of file main.cpp.

```
85                                          {
86     BTNode* u = rootNode; // Current node of interest
87     BTNode* prev = NULL;  // Previously looked at node
88     BTNode* next;         // The next node to look at
89
90     while(u != NULL) {
91         cout « "Traversing node " « u->nodeNum() « endl;
92         if(prev ==  u->parent) {
93             if(u->right != NULL) {
```

```
94                  next = u->right;
95              } else if(u->left != NULL) {
96                  next = u->left;
97              } else {
98                  next = u->parent;
99              }
100         } else if(prev == u->right) {
101             if(u->left != NULL){
102                 next = u->left;
103             } else {
104                 next = u->parent;
105             }
106         } else {
107             next = u->parent;
108         }
109         prev = u;
110         u = next;
111     }
112 }
```

4.2.2.6 traverse()

```
void traverse (
              BTNode * rootNode )
```

Traverses all the nodes in a binary tree.

Parameters

A	pointer to the root node of interest

Definition at line 69 of file main.cpp.

```
69                                  {
70     if(rootNode == NULL) {
71         cout « "reached NULL" « endl;
72         return;
73     }
74     cout « "Traversing node " « rootNode->nodeNum() « endl;
75     traverse(rootNode->right);
76     traverse(rootNode->left);
77 }
```

Index