HW_13_Heaps

1.0.0

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

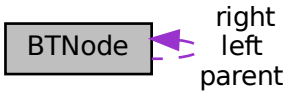
Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 BTNode Class Reference

Collaboration diagram for BTNode:



Public Member Functions

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

Public Attributes

- BTNode * left
- BTNode * right
- BTNode * parent

3.1.1 Detailed Description

Binary Tree Node

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

Definition at line 19 of file main.cpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BTNode()

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

BTNode constructor

Definition at line 28 of file main.cpp.

```
28                          {
29       cout « "name = " « name « endl;
30       left = NULL;
31       right = NULL;
32       parent = NULL;
33       objName = name++;
34       data = dataVal;
35   }
```

3.1.3 Member Function Documentation

3.1.3.1 nodeData()

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

This reports the node's data

Definition at line 47 of file main.cpp.

```
47                      {
48         return(data);
49     }
```

3.1.3.2 nodeName()

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

This reports the node's name

Definition at line 40 of file main.cpp.

```
40                       {
41         return(objName);
42     }
```

3.1.4 Member Data Documentation

3.1.4.1 left

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

Definition at line 21 of file main.cpp.

3.1.4.2 parent

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

Definition at line 23 of file main.cpp.

3.1.4.3 right

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

Definition at line 22 of file main.cpp.

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

- /home/scott/CPTR227/HW13_Heaps/src/main.cpp

Chapter 4

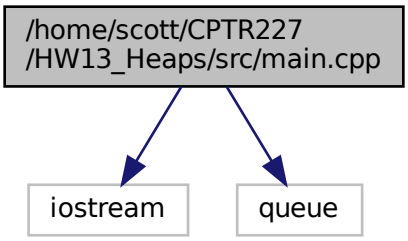
File Documentation

4.1 /home/scott/CPTR227/HW13_Heaps/src/main.cpp File Reference

This is a test of CMake, doxygen, and GitHub.

```
#include <iostream>
#include <queue>
```

Include dependency graph for main.cpp:



Classes

- class BTNode

Functions

- BTNode * addNode (BTNode *rootNode, BTNode *n)
- BTNode * addNode (BTNode *rootNode, int dataval)
- BTNode * merge (BTNode *h1, BTNode *h2)
- bool add (int x, BTNode *&rootNode)
- int remove (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 test of CMake, doxygen, and GitHub.

This is the long brief at the top of main.cpp.

Author

Scott Gillis

Date

3/30/2020

4.1.2 Function Documentation

4.1.2.1 add()

```
bool add (
            int x,
            BTNode *& rootNode )
```

Definition at line 134 of file main.cpp.

```
134                                     {
135     BTNode *u = new BTNode(x);
136     u->left = u->right = u->parent = NULL;
137     rootNode = merge(u, rootNode);
138     rootNode->parent = NULL;
139     //n++;
140     return true;
141 }
```

4.1.2.2 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 67 of file main.cpp.

```
67                                                  {
68     BTNode* prev = NULL;
69     BTNode* w = rootNode;
70     if(rootNode == NULL) { // starting an empty tree
71         rootNode = n;
72     } else {
73         // Find the node n belongs under, prev, n's new parent
74         while(w != NULL) {
75             prev = w;
76             if(n->nodeData() < w->nodeData()){
77                 w = w->left;
78             } else if(n->nodeData() > w->nodeData()) {
79                 w = w->right;
80             } else { // data already in the tree
81                 return(NULL);
82             }
83         }
84         // now prev should contain the node that should be n's parent
85         // Add n to prev
86         if(n->nodeData() < prev->nodeData()) {
87             prev->left = n;
88         } else {
89             prev->right = n;
90         }
91     }
92     return(rootNode);
93 }
```

4.1.2.3 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 103 of file main.cpp.

```
103                                                  {
104     BTNode* newNode = new BTNode(dataval);
105     if(addNode(rootNode, newNode) == NULL) {
106         cout « dataval « " already in tree" « endl;
107     } else {
108         cout « dataval « " succesfully added" « endl;
109     }
110     return(rootNode);
111 }
```

4.1.2.4 main()

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

Definition at line 192 of file main.cpp.

```
192                         {
193 BTNode* rootNode = new BTNode(5); // pointer to the root node  ;
194 add(3, rootNode);
195 printBT(rootNode);
196 add(10, rootNode);
197 printBT(rootNode);
198 add(15, rootNode);
199 printBT(rootNode);
200 add(32, rootNode);
201 printBT(rootNode);
202 add(8, rootNode);
203 printBT(rootNode);
204 remove(rootNode);
205 cout « "value removed: " « remove(rootNode) « endl;
206 printBT(rootNode);
207 remove(rootNode);
208 cout « "value removed: " « remove(rootNode) « endl;
209 printBT(rootNode);
210 add(38,rootNode);
211 add(198, rootNode);
212 add(65,rootNode);
213 remove(rootNode);
214 printBT(rootNode);
215
216 }
```

4.1.2.5 merge()

```
BTNode* merge (
              BTNode * h1,
              BTNode * h2 )
```

Definition at line 113 of file main.cpp.

```
113                                       {
114 if (h1 == NULL)
115     return h2;
116 if (h2 == NULL)
117     return h1;
118 if (h1->nodeData() > h2->nodeData()) //if h1 data is greater than h2 data
119     return merge(h2, h1);
120
121 // now we know h1->x <= h2->x
122 if(rand() % 2) {
123     h1->left = merge(h1->left, h2);
124     if (h1->left != NULL)
125         h1->left->parent = h1;
126 } else {
127     h1->right = merge(h1->right, h2);
128     if (h1->right != NULL)
129     h1->right->parent = h1;
130 }
131 return h1;
132 }
```

4.1.2.6 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 187 of file main.cpp.

```
188 {
189     printBT("", node, false);
190 }
```

4.1.2.7 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 164 of file main.cpp.

```
165 {
166     if( node != NULL )
167     {
168         cout « prefix;
169 
170         cout « (isLeft ? "L--" : "R--" );
171 
172         // print the value of the node
173         //cout « node->nodeName() « ':' « node->nodeData() « std::endl;
174         cout « node->nodeData() « std::endl;
175 
176         // enter the next tree level - left and right branch
177         printBT( prefix + (isLeft ? "|   " : "    "), node->left, true);
178         printBT( prefix + (isLeft ? "|   " : "    "), node->right, false);
179     }
180 }
```

4.1.2.8 remove()

```
int remove (
            BTNode *& rootNode )
```

Definition at line 143 of file main.cpp.

```
143                                 {
144     int x = rootNode->nodeData();
145     BTNode *tmp = rootNode;
146     rootNode = merge(rootNode->left, rootNode->right);
147     delete tmp;
148     if (rootNode != NULL)
149         rootNode->parent = NULL;
150     //n--;
151     return x;
152 }
```

Index