

HW_012

0.3.0

Generated by Doxygen 1.8.17

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 Node Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	5
3.1.2.1 color	6
3.1.2.2 data	6
3.1.2.3 left	6
3.1.2.4 parent	6
3.1.2.5 right	6
3.2 RBTree Class Reference	7
3.2.1 Detailed Description	7
3.2.2 Constructor & Destructor Documentation	7
3.2.2.1 RBTree()	7
3.2.3 Member Function Documentation	7
3.2.3.1 deleteNode()	8
3.2.3.2 getRoot()	8
3.2.3.3 inorder()	8
3.2.3.4 insert()	8
3.2.3.5 leftRotate()	9
3.2.3.6 maximum()	9
3.2.3.7 minimum()	10
3.2.3.8 postorder()	10
3.2.3.9 predecessor()	10
3.2.3.10 preorder()	10
3.2.3.11 prettyPrint()	11
3.2.3.12 rightRotate()	11
3.2.3.13 searchTree()	11
3.2.3.14 successor()	12
4 File Documentation	13
4.1 /home/scott/CPTR227/HW_012-Red-Black-Tree/src/main.cpp File Reference	13
4.1.1 Typedef Documentation	14
4.1.1.1 NodePtr	14
4.1.2 Function Documentation	14
4.1.2.1 main()	14
Index	15

Chapter 1

Class Index

1.1 Class List

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

Node	5
RBTREE	7

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

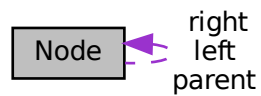
/home/scott/CPTR227/HW_012-Red-Black-Tree/src/[main.cpp](#) 13

Chapter 3

Class Documentation

3.1 Node Struct Reference

Collaboration diagram for Node:



Public Attributes

- int `data`
- `Node *` `parent`
- `Node *` `left`
- `Node *` `right`
- int `color`

3.1.1 Detailed Description

Definition at line 12 of file main.cpp.

3.1.2 Member Data Documentation

3.1.2.1 color

```
int Node::color
```

Definition at line 17 of file main.cpp.

3.1.2.2 data

```
int Node::data
```

Definition at line 13 of file main.cpp.

3.1.2.3 left

```
Node* Node::left
```

Definition at line 15 of file main.cpp.

3.1.2.4 parent

```
Node* Node::parent
```

Definition at line 14 of file main.cpp.

3.1.2.5 right

```
Node* Node::right
```

Definition at line 16 of file main.cpp.

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

- [/home/scott/CPTR227/HW_012-Red-Black-Tree/src/main.cpp](#)

3.2 RBTREE Class Reference

Public Member Functions

- [RBTREE](#) ()
- void [preorder](#) ()
- void [inorder](#) ()
- void [postorder](#) ()
- [NodePtr](#) [searchTree](#) (int k)
- [NodePtr](#) [minimum](#) ([NodePtr](#) node)
- [NodePtr](#) [maximum](#) ([NodePtr](#) node)
- [NodePtr](#) [successor](#) ([NodePtr](#) x)
- [NodePtr](#) [predecessor](#) ([NodePtr](#) x)
- void [leftRotate](#) ([NodePtr](#) x)
- void [rightRotate](#) ([NodePtr](#) x)
- void [insert](#) (int key)
- [NodePtr](#) [getRoot](#) ()
- void [deleteNode](#) (int data)
- void [prettyPrint](#) ()

3.2.1 Detailed Description

Definition at line 23 of file main.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 RBTREE()

```
RBTREE::RBTREE ( ) [inline]
```

Definition at line 278 of file main.cpp.

```
278     {
279         TNULL = new Node;
280         TNULL->color = 0;
281         TNULL->left = nullptr;
282         TNULL->right = nullptr;
283         root = TNULL;
284     }
```

3.2.3 Member Function Documentation

3.2.3.1 deleteNode()

```
void RBTre::deleteNode (
    int data ) [inline]
```

Definition at line 454 of file main.cpp.

```
454     {
455         deleteNodeHelper(this->root, data);
456     }
```

3.2.3.2 getRoot()

```
NodePtr RBTre::getRoot ( ) [inline]
```

Definition at line 449 of file main.cpp.

```
449     {
450         return this->root;
451     }
```

3.2.3.3 inorder()

```
void RBTre::inorder ( ) [inline]
```

Definition at line 294 of file main.cpp.

```
294     {
295         inOrderHelper(this->root);
296     }
```

3.2.3.4 insert()

```
void RBTre::insert (
    int key ) [inline]
```

Definition at line 403 of file main.cpp.

```
403     {
404         // Ordinary Binary Search Insertion
405         NodePtr node = new Node;
406         node->parent = nullptr;
407         node->data = key;
408         node->left = TNULL;
409         node->right = TNULL;
410         node->color = 1; // new node must be red
411
412         NodePtr y = nullptr;
413         NodePtr x = this->root;
414
415         while (x != TNULL) {
416             y = x;
417             if (node->data < x->data) {
418                 x = x->left;
419             } else {
420                 x = x->right;
421             }
422         }
423
424         // y is parent of x
425         node->parent = y;
426         if (y == nullptr) {
```

```

427         root = node;
428     } else if (node->data < y->data) {
429         y->left = node;
430     } else {
431         y->right = node;
432     }
433
434     // if new node is a root node, simply return
435     if (node->parent == nullptr) {
436         node->color = 0;
437         return;
438     }
439
440     // if the grandparent is null, simply return
441     if (node->parent->parent == nullptr) {
442         return;
443     }
444
445     // Fix the tree
446     fixInsert(node);
447 }

```

3.2.3.5 leftRotate()

```

void RBTree::leftRotate (
    NodePtr x ) [inline]

```

Definition at line 364 of file main.cpp.

```

364     {
365         NodePtr y = x->right;
366         x->right = y->left;
367         if (y->left != TNULL) {
368             y->left->parent = x;
369         }
370         y->parent = x->parent;
371         if (x->parent == nullptr) {
372             this->root = y;
373         } else if (x == x->parent->left) {
374             x->parent->left = y;
375         } else {
376             x->parent->right = y;
377         }
378         y->left = x;
379         x->parent = y;
380     }

```

3.2.3.6 maximum()

```

NodePtr RBTree::maximum (
    NodePtr node ) [inline]

```

Definition at line 319 of file main.cpp.

```

319     {
320         while (node->right != TNULL) {
321             node = node->right;
322         }
323         return node;
324     }

```

3.2.3.7 minimum()

```
NodePtr RBTREE::minimum (
    NodePtr node ) [inline]
```

Definition at line 311 of file main.cpp.

```
311     {
312         while (node->left != TNULL) {
313             node = node->left;
314         }
315         return node;
316     }
```

3.2.3.8 postorder()

```
void RBTREE::postorder ( ) [inline]
```

Definition at line 300 of file main.cpp.

```
300     {
301         postOrderHelper(this->root);
302     }
```

3.2.3.9 predecessor()

```
NodePtr RBTREE::predecessor (
    NodePtr x ) [inline]
```

Definition at line 346 of file main.cpp.

```
346     {
347         // if the left subtree is not null,
348         // the predecessor is the rightmost node in the
349         // left subtree
350         if (x->left != TNULL) {
351             return maximum(x->left);
352         }
353         NodePtr y = x->parent;
354         while (y != TNULL && x == y->left) {
355             x = y;
356             y = y->parent;
357         }
358         return y;
359     }
360
361 }
```

3.2.3.10 preorder()

```
void RBTREE::preorder ( ) [inline]
```

Definition at line 288 of file main.cpp.

```
288     {
289         preOrderHelper(this->root);
290     }
```

3.2.3.11 prettyPrint()

```
void RBTREE::prettyPrint ( ) [inline]
```

Definition at line 459 of file main.cpp.

```
459         {
460             if (root) {
461                 printHelper(this->root, "", true);
462             }
463         }
```

3.2.3.12 rightRotate()

```
void RBTREE::rightRotate (
    NodePtr x ) [inline]
```

Definition at line 383 of file main.cpp.

```
383         {
384             NodePtr y = x->left;
385             x->left = y->right;
386             if (y->right != TNULL) {
387                 y->right->parent = x;
388             }
389             y->parent = x->parent;
390             if (x->parent == nullptr) {
391                 this->root = y;
392             } else if (x == x->parent->right) {
393                 x->parent->right = y;
394             } else {
395                 x->parent->left = y;
396             }
397             y->right = x;
398             x->parent = y;
399         }
```

3.2.3.13 searchTree()

```
NodePtr RBTREE::searchTree (
    int k ) [inline]
```

Definition at line 306 of file main.cpp.

```
306         {
307             return searchTreeHelper(this->root, k);
308         }
```

3.2.3.14 successor()

```
NodePtr RBTREE::successor (
    NodePtr x ) [inline]
```

Definition at line 327 of file main.cpp.

```
327     {
328         // if the right subtree is not null,
329         // the successor is the leftmost node in the
330         // right subtree
331         if (x->right != TNULL) {
332             return minimum(x->right);
333         }
334         // else it is the lowest ancestor of x whose
335         // left child is also an ancestor of x.
336         NodePtr y = x->parent;
337         while (y != TNULL && x == y->right) {
338             x = y;
339             y = y->parent;
340         }
341         return y;
342     }
343 }
```

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

- /home/scott/CPTR227/HW_012-Red-Black-Tree/src/main.cpp

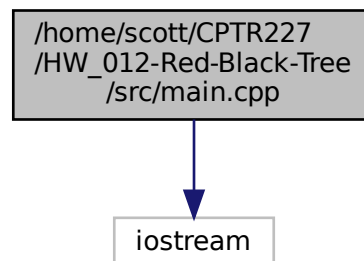
Chapter 4

File Documentation

4.1 /home/scott/CPTR227/HW_012-Red-Black-Tree/src/main.cpp File Reference

```
#include <iostream>
```

Include dependency graph for main.cpp:



Classes

- struct `Node`
- class `RBTree`

Typedefs

- typedef `Node * NodePtr`

Functions

- int `main ()`

4.1.1 Typedef Documentation

4.1.1.1 NodePtr

```
typedef Node* NodePtr
```

Definition at line 20 of file main.cpp.

4.1.2 Function Documentation

4.1.2.1 main()

```
int main ( )
```

Definition at line 467 of file main.cpp.

```
467     {  
468         RBTREE bst;  
469         bst.insert(1);  
470         bst.insert(11);  
471         bst.insert(521312);  
472         bst.insert(2);  
473         bst.insert(1302134);  
474         bst.insert(3);  
475         bst.insert(402141);  
476         bst.insert(81240);  
477         bst.deleteNode(2);  
478         bst.prettyPrint();  
479         return 0;  
480     }
```

Index

/home/scott/CPTR227/HW_012-Red-Black-Tree/src/main.cpp, [prettyPrint](#)
[13](#) [RBTree, 10](#)

color
 Node, [5](#)

data
 Node, [6](#)

deleteNode
 RBTree, [7](#)

getRoot
 RBTree, [8](#)

inorder
 RBTree, [8](#)

insert
 RBTree, [8](#)

left
 Node, [6](#)

leftRotate
 RBTree, [9](#)

main
 main.cpp, [14](#)

main.cpp
 main, [14](#)
 NodePtr, [14](#)

maximum
 RBTree, [9](#)

minimum
 RBTree, [9](#)

Node, [5](#)
 color, [5](#)
 data, [6](#)
 left, [6](#)
 parent, [6](#)
 right, [6](#)

NodePtr
 main.cpp, [14](#)

parent
 Node, [6](#)

postorder
 RBTree, [10](#)

predecessor
 RBTree, [10](#)

preorder
 RBTree, [10](#)

RBTree, [7](#)
 deleteNode, [7](#)
 getRoot, [8](#)
 inorder, [8](#)
 insert, [8](#)
 leftRotate, [9](#)
 maximum, [9](#)
 minimum, [9](#)
 postorder, [10](#)
 predecessor, [10](#)
 preorder, [10](#)
 prettyPrint, [10](#)
 RBTree, [7](#)
 rightRotate, [11](#)
 searchTree, [11](#)
 successor, [11](#)

right
 Node, [6](#)

rightRotate
 RBTree, [11](#)

searchTree
 RBTree, [11](#)

successor
 RBTree, [11](#)