

aardvark

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/addis/RED-BLACK-TREE-S/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:

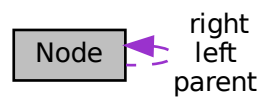
/home/addis/RED-BLACK-TREE-S/src/[main.cpp](#) 13

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 11 of file main.cpp.

3.1.2 Member Data Documentation

3.1.2.1 color

```
int Node::color
```

Definition at line 16 of file main.cpp.

3.1.2.2 data

```
int Node::data
```

Definition at line 12 of file main.cpp.

3.1.2.3 left

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

Definition at line 14 of file main.cpp.

3.1.2.4 parent

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

Definition at line 13 of file main.cpp.

3.1.2.5 right

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

Definition at line 15 of file main.cpp.

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

- [/home/addis/RED-BLACK-TREE-S/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 22 of file main.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 RBTREE()

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

Definition at line 277 of file main.cpp.

```
277     {
278         TNULL = new Node;
279         TNULL->color = 0;
280         TNULL->left = nullptr;
281         TNULL->right = nullptr;
282         root = TNULL;
283     }
```

3.2.3 Member Function Documentation

3.2.3.1 deleteNode()

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

Definition at line 453 of file main.cpp.

```
453     {
454         deleteNodeHelper(this->root, data);
455     }
```

3.2.3.2 getRoot()

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

Definition at line 448 of file main.cpp.

```
448     {
449         return this->root;
450     }
```

3.2.3.3 inorder()

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

Definition at line 293 of file main.cpp.

```
293     {
294         inOrderHelper(this->root);
295     }
```

3.2.3.4 insert()

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

Definition at line 402 of file main.cpp.

```
402     {
403         // Ordinary Binary Search Insertion
404         NodePtr node = new Node;
405         node->parent = nullptr;
406         node->data = key;
407         node->left = TNULL;
408         node->right = TNULL;
409         node->color = 1; // new node must be red
410
411         NodePtr y = nullptr;
412         NodePtr x = this->root;
413
414         while (x != TNULL) {
415             y = x;
416             if (node->data < x->data) {
417                 x = x->left;
418             } else {
419                 x = x->right;
420             }
421         }
422
423         // y is parent of x
424         node->parent = y;
425         if (y == nullptr) {
```

```

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

```

3.2.3.5 leftRotate()

```

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

```

Definition at line 363 of file main.cpp.

```

363     {
364         NodePtr y = x->right;
365         x->right = y->left;
366         if (y->left != TNULL) {
367             y->left->parent = x;
368         }
369         y->parent = x->parent;
370         if (x->parent == nullptr) {
371             this->root = y;
372         } else if (x == x->parent->left) {
373             x->parent->left = y;
374         } else {
375             x->parent->right = y;
376         }
377         y->left = x;
378         x->parent = y;
379     }

```

3.2.3.6 maximum()

```

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

```

Definition at line 318 of file main.cpp.

```

318     {
319         while (node->right != TNULL) {
320             node = node->right;
321         }
322         return node;
323     }

```

3.2.3.7 minimum()

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

Definition at line 310 of file main.cpp.

```
310     {
311         while (node->left != TNULL) {
312             node = node->left;
313         }
314         return node;
315     }
```

3.2.3.8 postorder()

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

Definition at line 299 of file main.cpp.

```
299     {
300         postOrderHelper(this->root);
301     }
```

3.2.3.9 predecessor()

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

Definition at line 345 of file main.cpp.

```
345     {
346         // if the left subtree is not null,
347         // the predecessor is the rightmost node in the
348         // left subtree
349         if (x->left != TNULL) {
350             return maximum(x->left);
351         }
352
353         NodePtr y = x->parent;
354         while (y != TNULL && x == y->left) {
355             x = y;
356             y = y->parent;
357         }
358
359         return y;
360     }
```

3.2.3.10 preorder()

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

Definition at line 287 of file main.cpp.

```
287     {
288         preOrderHelper(this->root);
289     }
```

3.2.3.11 prettyPrint()

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

Definition at line 458 of file main.cpp.

```
458         {
459             if (root) {
460                 printHelper(this->root, "", true);
461             }
462         }
```

3.2.3.12 rightRotate()

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

Definition at line 382 of file main.cpp.

```
382         {
383             NodePtr y = x->left;
384             x->left = y->right;
385             if (y->right != TNULL) {
386                 y->right->parent = x;
387             }
388             y->parent = x->parent;
389             if (x->parent == nullptr) {
390                 this->root = y;
391             } else if (x == x->parent->right) {
392                 x->parent->right = y;
393             } else {
394                 x->parent->left = y;
395             }
396             y->right = x;
397             x->parent = y;
398         }
```

3.2.3.13 searchTree()

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

Definition at line 305 of file main.cpp.

```
305         {
306             return searchTreeHelper(this->root, k);
307         }
```

3.2.3.14 successor()

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

Definition at line 326 of file main.cpp.

```
326 {
327     // if the right subtree is not null,
328     // the successor is the leftmost node in the
329     // right subtree
330     if (x->right != TNULL) {
331         return minimum(x->right);
332     }
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 }
```

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

- [/home/addis/RED-BLACK-TREE-S/src/main.cpp](#)

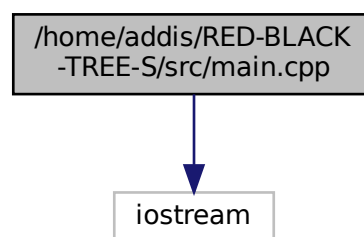
Chapter 4

File Documentation

4.1 /home/addis/RED-BLACK-TREE-S/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 19 of file main.cpp.

4.1.2 Function Documentation

4.1.2.1 main()

```
int main ( )
```

Definition at line 466 of file main.cpp.

```
466     {
467         RBTREE bst;
468         bst.insert(8);
469         bst.insert(18);
470         bst.insert(5);
471         bst.insert(15);
472         bst.insert(17);
473         bst.insert(25);
474         bst.insert(40);
475         bst.insert(80);
476         bst.deleteNode(25);
477         bst.prettyPrint();
478
479         bst.insert(9);
480         bst.insert(19);
481         bst.insert(6);
482         bst.insert(16);
483         bst.insert(18);
484         bst.insert(26);
485         bst.insert(41);
486         bst.insert(81);
487         bst.deleteNode(26);
488         bst.prettyPrint();
489
490         bst.insert(7);
491         bst.insert(17);
492         bst.insert(4);
493         bst.insert(14);
494         bst.insert(16);
495         bst.insert(24);
496         bst.insert(39);
497         bst.insert(79);
498         bst.deleteNode(24);
499         bst.prettyPrint();
500
501         bst.insert(10);
502         bst.insert(20);
503         bst.insert(7);
504         bst.insert(17);
505         bst.insert(19);
506         bst.insert(27);
507         bst.insert(42);
508         bst.insert(82);
509         bst.deleteNode(27);
510         bst.prettyPrint();
511
512         bst.insert(6);
513         bst.insert(16);
514         bst.insert(3);
515         bst.insert(13);
516         bst.insert(15);
517         bst.insert(23);
518         bst.insert(38);
519         bst.insert(79);
520         bst.deleteNode(23);
521         bst.prettyPrint();
522         return 0;
523 }
```

Index

/home/addis/RED-BLACK-TREE-S/src/main.cpp, [13](#)

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](#)

prettyPrint

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](#)