aardvark

0.3.0

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

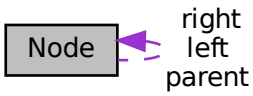
Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 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/bona/CPTR227/-Red-Black-Trees/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 RBTree::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 RBTree::getRoot ( )  [inline]
```

Definition at line 449 of file main.cpp.

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

3.2.3.3 inorder()

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

Definition at line 294 of file main.cpp.

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

3.2.3.4 insert()

```
void RBTree::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
354         NodePtr y = x->parent;
355         while (y != TNULL && x == y->left) {
356             x = y;
357             y = y->parent;
358         }
359
360         return y;
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 
335         // else it is the lowest ancestor of x whose
336         // left child is also an ancestor of x.
337         NodePtr y = x->parent;
338         while (y != TNULL && x == y->right) {
339             x = y;
340             y = y->parent;
341         }
342         return y;
343     }
```

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

- /home/bona/CPTR227/-Red-Black-Trees/src/main.cpp

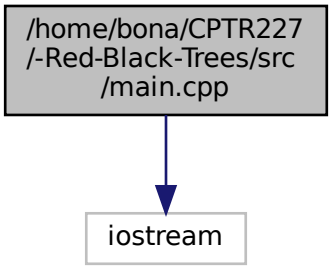
Chapter 4

File Documentation

4.1 /home/bona/CPTR227/-Red-Black-Trees/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(8);
470     bst.insert(18);
471     bst.insert(5);
472     bst.insert(15);
473     bst.insert(17);
474     bst.insert(25);
475     bst.insert(40);
476     bst.insert(80);
477     bst.deleteNode(25);
478     bst.prettyPrint();
479     bst.insert(4);
480     bst.insert(5);
481     bst.insert(9);
482     bst.insert(15);
483     bst.insert(11);
484     bst.insert(22);
485     bst.insert(42);
486     bst.insert(81);
487     bst.deleteNode(21);
488     bst.prettyPrint();
489     bst.insert(1);
490     bst.insert(12);
491     bst.insert(9);
492     bst.insert(13);
493     bst.insert(19);
494     bst.insert(11);
495     bst.insert(44);
496     bst.insert(84);
497     bst.deleteNode(29);
498     bst.prettyPrint();
499     bst.insert(3);
500     bst.insert(17);
501     bst.insert(8);
502     bst.insert(14);
503     bst.insert(18);
504     bst.insert(26);
505     bst.insert(49);
506     bst.insert(85);
507     bst.deleteNode(22);
508     bst.prettyPrint();
509     bst.insert(7);
510     bst.insert(16);
511     bst.insert(3);
512     bst.insert(14);
513     bst.insert(18);
514     bst.insert(27);
515     bst.insert(45);
516     bst.insert(88);
```

```
517     bst.deleteNode(23);
518     bst.prettyPrint();
519     bst.insert(2);
520     bst.insert(13);
521     bst.insert(9);
522     bst.insert(12);
523     bst.insert(22);
524     bst.insert(11);
525     bst.insert(48);
526     bst.insert(89);
527     bst.deleteNode(21);
528     bst.prettyPrint();
529     return 0;
530
531 }
```

Index