RedBlackTrees

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

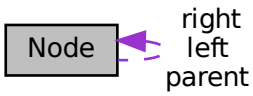
Here is a list of all files with brief descriptions:

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 14 of file main.cpp.

3.1.2 Member Data Documentation

3.1.2.1 color

```
int Node::color
```

Definition at line 19 of file main.cpp.

3.1.2.2 data

```
int Node::data
```

Definition at line 15 of file main.cpp.

3.1.2.3 left

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

Definition at line 17 of file main.cpp.

3.1.2.4 parent

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

Definition at line 16 of file main.cpp.

3.1.2.5 right

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

Definition at line 18 of file main.cpp.

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

- /home/benson/CPTR227/RedBlackTrees/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 25 of file main.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 RBTree()

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

Definition at line 280 of file main.cpp.

```
280         {
281 TNULL = new Node;
282 TNULL->color = 0;
283 TNULL->left = nullptr;
284 TNULL->right = nullptr;
285 root = TNULL;
286 }
```

3.2.3 Member Function Documentation

3.2.3.1 deleteNode()

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

Definition at line 456 of file main.cpp.

```
456                         {
457 deleteNodeHelper(this->root, data);
458 }
```

3.2.3.2 getRoot()

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

Definition at line 451 of file main.cpp.

```
451                  {
452 return this->root;
453 }
```

3.2.3.3 inorder()

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

Definition at line 296 of file main.cpp.

```
296                {
297 inOrderHelper(this->root);
298 }
```

3.2.3.4 insert()

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

Definition at line 405 of file main.cpp.

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

```
429 root = node;
430 } else if (node->data < y->data) {
431 y->left = node;
432 } else {
433 y->right = node;
434 }
435
436 // if new node is a root node, simply return
437 if (node->parent == nullptr){
438 node->color = 0;
439 return;
440 }
441
442 // if the grandparent is null, simply return
443 if (node->parent->parent == nullptr) {
444 return;
445 }
446
447 // Fix the tree
448 fixInsert(node);
449 }
```

3.2.3.5 leftRotate()

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

Definition at line 366 of file main.cpp.

```
366                              {
367 NodePtr y = x->right;
368 x->right = y->left;
369 if (y->left != TNULL) {
370 y->left->parent = x;
371 }
372 y->parent = x->parent;
373 if (x->parent == nullptr) {
374 this->root = y;
375 } else if (x == x->parent->left) {
376 x->parent->left = y;
377 } else {
378 x->parent->right = y;
379 }
380 y->left = x;
381 x->parent = y;
382 }
```

3.2.3.6 maximum()

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

Definition at line 321 of file main.cpp.

```
321                                 {
322 while (node->right != TNULL) {
323 node = node->right;
324 }
325 return node;
326 }
```

3.2.3.7 minimum()

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

Definition at line 313 of file main.cpp.

```
313                                 {
314 while (node->left != TNULL) {
315 node = node->left;
316 }
317 return node;
318 }
```

3.2.3.8 postorder()

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

Definition at line 302 of file main.cpp.

```
302                  {
303 postOrderHelper(this->root);
304 }
```

3.2.3.9 predecessor()

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

Definition at line 348 of file main.cpp.

```
348                                   {
349 // if the left subtree is not null,
350 // the predecessor is the rightmost node in the
351 // left subtree
352 if (x->left != TNULL) {
353 return maximum(x->left);
354 }
355
356 NodePtr y = x->parent;
357 while (y != TNULL && x == y->left) {
358 x = y;
359 y = y->parent;
360 }
361
362 return y;
363 }
```

3.2.3.10 preorder()

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

Definition at line 290 of file main.cpp.

```
290                  {
291 preOrderHelper(this->root);
292 }
```

3.2.3.11 prettyPrint()

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

Definition at line 461 of file main.cpp.

```
461                     {
462     if (root) {
463      printHelper(this->root, "", true);
464     }
465 }
```

3.2.3.12 rightRotate()

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

Definition at line 385 of file main.cpp.

```
385                              {
386 NodePtr y = x->left;
387 x->left = y->right;
388 if (y->right != TNULL) {
389 y->right->parent = x;
390 }
391 y->parent = x->parent;
392 if (x->parent == nullptr) {
393 this->root = y;
394 } else if (x == x->parent->right) {
395 x->parent->right = y;
396 } else {
397 x->parent->left = y;
398 }
399 y->right = x;
400 x->parent = y;
401 }
```

3.2.3.13 searchTree()

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

Definition at line 308 of file main.cpp.

```
308                            {
309 return searchTreeHelper(this->root, k);
310 }
```

3.2.3.14 successor()

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

Definition at line 329 of file main.cpp.

```
329                             {
330 // if the right subtree is not null,
331 // the successor is the leftmost node in the
332 // right subtree
333 if (x->right != TNULL) {
334 return minimum(x->right);
335 }
336
337 // else it is the lowest ancestor of x whose
338 // left child is also an ancestor of x.
339 NodePtr y = x->parent;
340 while (y != TNULL && x == y->right) {
341 x = y;
342 y = y->parent;
343 }
344 return y;
345 }
```

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

- /home/benson/CPTR227/RedBlackTrees/src/main.cpp

Chapter 4

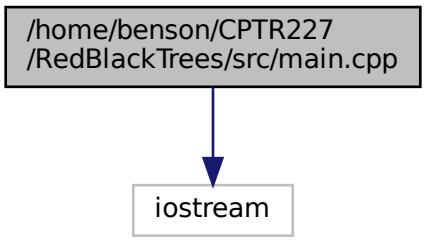
File Documentation

4.1 /home/benson/CPTR227/RedBlackTrees/src/main.cpp File Reference

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

```
#include <iostream>
```

Include dependency graph for main.cpp:



Classes

- struct Node
- class RBTree

Typedefs

- typedef Node * NodePtr

Functions

- int main ()

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

Benson Nyakango

Date

3/24/20

4.1.2 Typedef Documentation

4.1.2.1 NodePtr

```
typedef Node* NodePtr
```

Definition at line 22 of file main.cpp.

4.1.3 Function Documentation

4.1.3.1 main()

```
int main ( )
```

Definition at line 469 of file main.cpp.

```
469           {
470 RBTree bst;
471 bst.insert(8);
472 bst.insert(18);
473 bst.insert(5);
474 bst.insert(16);
475 bst.insert(17);
476 bst.insert(25);
477 bst.insert(40);
478 bst.insert(80);
479 bst.deleteNode(25);
480 bst.prettyPrint();
481     bst.insert(4);
482 bst.insert(12);
483 bst.insert(9);
484 bst.insert(15);
485 bst.insert(11);
486 bst.insert(22);
487 bst.insert(44);
488 bst.insert(81);
489 bst.deleteNode(21);
490 bst.prettyPrint();
491     bst.insert(1);
492 bst.insert(12);
493 bst.insert(9);
```

```
494 bst.insert(13);
495 bst.insert(19);
496 bst.insert(11);
497 bst.insert(44);
498 bst.insert(84);
499 bst.deleteNode(29);
500 bst.prettyPrint();
501     bst.insert(3);
502 bst.insert(17);
503 bst.insert(8);
504 bst.insert(14);
505 bst.insert(18);
506 bst.insert(26);
507 bst.insert(49);
508 bst.insert(85);
509 bst.deleteNode(22);
510 bst.prettyPrint();
511     bst.insert(7);
512 bst.insert(16);
513 bst.insert(3);
514 bst.insert(14);
515 bst.insert(18);
516 bst.insert(27);
517 bst.insert(45);
518 bst.insert(88);
519 bst.deleteNode(23);
520 bst.prettyPrint();
521     bst.insert(2);
522 bst.insert(13);
523 bst.insert(9);
524 bst.insert(12);
525 bst.insert(22);
526 bst.insert(11);
527 bst.insert(48);
528 bst.insert(89);
529 bst.deleteNode(21);
530 bst.prettyPrint();
531 return 0;
532
533 }
```

Index