#include<stdio.h>

#include<stdlib.h>

enum COLOR {Red,Black};

typedef struct tree\_node

{

int data;

struct tree\_node \*right;

struct tree\_node \*left;

struct tree\_node \*parent;

enum COLOR color;

}

tree\_node;

typedef struct red\_black\_tree

{

tree\_node \*root;

tree\_node \*NIL;

}

red\_black\_tree;

tree\_node\* new\_tree\_node(int data)

{

tree\_node\* n = malloc(sizeof(tree\_node));

n->left = NULL;

n->right = NULL;

n->parent = NULL;

n->data = data;

n->color = Red;

return n;

}

red\_black\_tree\* new\_red\_black\_tree()

{

red\_black\_tree \*t = malloc(sizeof(red\_black\_tree));

tree\_node \*nil\_node = malloc(sizeof(tree\_node));

nil\_node->left = NULL;

nil\_node->right = NULL;

nil\_node->parent = NULL;

nil\_node->color = Black;

nil\_node->data = 0;

t->NIL = nil\_node;

t->root = t->NIL;

return t;

}

void left\_rotate(red\_black\_tree \*t, tree\_node \*x)

{

tree\_node \*y = x->right;

x->right = y->left;

if(y->left != t->NIL)

{

y->left->parent = x;

}

y->parent = x->parent;

if(x->parent == t->NIL)

{

// x is root

t->root = y;

}

else if(x == x->parent->left)

{

// x is left child

x->parent->left = y;

}

else { // x is right child

x->parent->right = y;

}

y->left = x;

x->parent = y;

}

void right\_rotate(red\_black\_tree \*t, tree\_node \*x)

{

tree\_node \*y = x->left;

x->left = y->right;

if(y->right != t->NIL)

{ //x is root

t->root = y;

}

else if(x == x->parent->right)

{

// xis left child

x->parent->right = y;

}

else { //x is right child

x->parent->left = y;

}

y->right = x;

x->parent = y;

}

void insertion\_fixup(red\_black\_tree \*t, tree\_node \*z)

{

while(z->parent->color == Red)

{

if(z->parent == z->parent->parent->left)

{

// z,parent is the left child

tree\_node \*y = z->parent->parent->right; //uncle of z

if(y->color == Red)

{

//case 1

z->parent->color = Black;

y->color = Black;

z->parent->parent->color = Red;

z = z->parent->parent;

}

else

{ //case 2 or case 3

if(z ==z->parent->right) { //case 2

z = z->parent; //marked z.parent as a new z

left\_rotate(t, z);

}

//case 3

z->parent->color = Black; //made parent black

z->parent->parent->color = Red; // made parent red

right\_rotate(t, z->parent->parent);

}

}

else { // z.parent is the right child

tree\_node \*y =z->parent->parent->left ; // uncle of z

if(y->color == Red) {

z->parent->color = Black;

y->color = Black;

z->parent->parent->color = Red;

z= z->parent->parent;

}

else {

if(z == z->parent->left)

{

z = z->parent ; //marked z parent as z new

right\_rotate(t, z);

}

z->parent->color = Black; // made parent black

z->parent->parent->color = Red; // made parent red

left\_rotate(t, z->parent->parent);

}

}

}

t->root->color = Black;

}

void insert(red\_black\_tree \*t, tree\_node \*z)

{

tree\_node\* y = t->NIL; //variable for the parent of the added node

tree\_node\* temp = t->root;

while(temp != t->NIL) {

y = temp ;

if(z->data < temp->data)

temp = temp->left;

else

temp = temp->right;

}

z->parent = y;

if(y == t-> NIL) { //newly added node is root

t->root = z;

}

else if(z->data < y->data) //data of child is less than its parent,left child

y->left = z;

else

y->right = z;

z->right = t->NIL;

z->left = t->NIL;

insertion\_fixup(t, z);

}

void inorder(red\_black\_tree \*t, tree\_node \*n)

{

if(n != t->NIL)

{

inorder(t, n->left);

printf("%d\n", n->data);

inorder(t, n->right);

}}

int main()

{

red\_black\_tree \*t = new\_red\_black\_tree();

tree\_node \*a, \*b, \*c, \*d, \*e, \*f, \*g, \*h, \*i, \*j, \*k, \*l, \*m;

a = new\_tree\_node(10);

b = new\_tree\_node(20);

c = new\_tree\_node(100);

d = new\_tree\_node(90);

e = new\_tree\_node(70);

f = new\_tree\_node(60);

g = new\_tree\_node(40);

h = new\_tree\_node(50);

i = new\_tree\_node(10);

j = new\_tree\_node(90);

k = new\_tree\_node(100);

l = new\_tree\_node(80);

m = new\_tree\_node(30);

insert(t, a);

insert(t, b);

insert(t, c);

insert(t, d);

insert(t, e);

insert(t, f);

insert(t, g);

insert(t, h);

insert(t, i);

insert(t, j);

insert(t, k);

insert(t, l);

insert(t, m);

inorder(t, t->root);

return 0;

}