#include <stdio.h>

#include <stdlib.h>

#include<mpi.h>

struct rbtNode

{ int key;

char color;

struct rbtNode \*left, \*right,\*parent;

}; struct rbtNode\* root = NULL;

void leftRotate(struct rbtNode \*x)

{ struct rbtNode \*y;

y = x->right; x->right = y->left;

if( y->left != NULL)

{ y->left->parent = x;

}

y->parent = x->parent;

if( x->parent == NULL)

{ root = y;

}

else if( (x->parent->left!=NULL) && (x->key == x->parent->left->key))

{ x->parent->left = y;

}

else x->parent->right = y;

y->left = x; x->parent = y; return;

}

void rightRotate(struct rbtNode \*y)

{ struct rbtNode \*x;

x = y->left; y->left = x->right;

if ( x->right != NULL)

{ x->right->parent = y;

}

x->parent = y->parent;

if( y->parent == NULL)

{ root = x;

}

else if((y->parent->left!=NULL)&& (y->key == y->parent->left->key))

{ y->parent->left = x;

}

else

y->parent->right = x;

x->right = y; y->parent = x;

return;

}

void color-insert(struct rbtNode \*z)

{ struct rbtNode \*y=NULL;

while ((z->parent != NULL) && (z->parent->color == 'r'))

{

if((z->parent->parent->left!=NULL)&&(z->parent->key==z->parent->parent->left->key))

{

if(z->parent->parent->right!=NULL)

y = z->parent->parent->right;

if ((y!=NULL) && (y->color == 'r'))

{ z->parent->color = 'b';

y->color = 'b';

z->parent->parent->color = 'r';

if(z->parent->parent!=NULL)

z = z->parent->parent;

}

else

{

if ((z->parent->right != NULL) && (z->key == z->parent->right->key))

{ z = z->parent;

leftRotate(z);

}

z->parent->color = 'b';

z->parent->parent->color = 'r';

rightRotate(z->parent->parent);

}

}

else

{

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

y = z->parent->parent->left;

if ((y!=NULL) && (y->color == 'r'))

{ z->parent->color = 'b';

y->color = 'b';

z->parent->parent->color = 'r';

if(z->parent->parent!=NULL)

z = z->parent->parent;

}

else

{

if ((z->parent->left != NULL) && (z->key == z->parent->left->key))

{ z = z->parent;

rightRotate(z);

}

z->parent->color = 'b';

z->parent->parent->color = 'r';

leftRotate(z->parent->parent);

}

}

} root->color = 'b';

}

void insert(int val)

{ struct rbtNode \*x, \*y;

struct rbtNode \*z = (struct rbtNode\*)malloc(sizeof(struct rbtNode));

z->key = val;

z->left = NULL;

z->right = NULL;

z->color = 'r';

x=root;

if ( root == NULL )

{ root = z;

root->color = 'b';

return;

}

while ( x != NULL)

{ y = x;

if ( z->key < x->key)

{ x = x->left;

}

else

x = x->right;

}

z->parent = y;

if ( y == NULL)

{ root = z;

}

else if( z->key < y->key )

{ y->left = z;

}

else

y->right = z;

color-insert(z);

return;

}

void inorderTree(struct rbtNode\* root)

{ struct rbtNode\* temp = root;

if (temp != NULL)

{ inorderTree(temp->left);

printf(" %d--%c ",temp->key,temp->color);

inorderTree(temp->right);

}

return;

}

void postorderTree(struct rbtNode\* root)

{ struct rbtNode\* temp = root;

if (temp != NULL)

{ postorderTree(temp->left);

postorderTree(temp->right);

printf(" %d--%c ",temp->key,temp->color);

}

return;

}

void traversal(struct rbtNode\* root)

{ if (root != NULL)

{ printf("root is %d-- %c",root->key,root->color);

printf("\nInorder tree traversal\n");

inorderTree(root);

printf("\npostorder tree traversal\n");

postorderTree(root);

}

return;

}

struct rbtNode\* min(struct rbtNode \*x)

{ while (x->left)

{ x = x->left;

}

return x;

}

struct rbtNode\* successor(struct rbtNode \*x)

{ struct rbtNode \*y;

if (x->right)

{ return min(x->right);

}

y = x->parent;

while (y && x == y->right)

{ x = y;

y = y->parent;

}

return y;

}

void color-delete(struct rbtNode \*x)

{ while (x != root && x->color == 'b')

{ struct rbtNode \*w = NULL;

if ((x->parent->left!=NULL) && (x == x->parent->left))

{ w = x->parent->right;

if ((w!=NULL) && (w->color == 'r'))

{ w->color = 'b';

x->parent->color = 'r';

leftRotate(x->parent);

w = x->parent->right;

}

if ((w!=NULL) && (w->right!=NULL) && (w->left!=NULL) && (w->left->color == 'b') && (w->right->color == 'b'))

{

w->color = 'r';

x = x->parent;

}

else if((w!=NULL) && (w->right->color == 'b'))

{ w->left->color = 'b';

w->color = 'r';

rightRotate(w);

w = x->parent->right;

}

if(w!=NULL)

{ w->color = x->parent->color;

x->parent->color = 'b';

w->right->color = 'b';

leftRotate(x->parent);

x = root;

}

}

else if(x->parent!=NULL)

{ w = x->parent->left;

if ((w!=NULL) && (w->color == 'r'))

{

w->color = 'b';

x->parent->color = 'r';

leftRotate(x->parent);

if(x->parent!=NULL)

w = x->parent->left;

}

if ((w!=NULL) && (w->right!=NULL) && (w->left!=NULL) && (w->right->color == 'b') && (w->left->color == 'b'))

{ x = x->parent;

}

else if((w!=NULL) && (w->right!=NULL) && (w->left!=NULL) && (w->left- >color == 'b'))

{ w->right->color = 'b';

w->color = 'r';

rightRotate(w);

w = x->parent->left;

}

if(x->parent!=NULL)

{ w->color = x->parent->color;

x->parent->color = 'b';

}

if(w->left!=NULL)

w->left->color = 'b';

if(x->parent !=NULL)

leftRotate(x->parent);

x = root;

}

} x->color = 'b';

}

struct rbtNode\* delete(int var)

{ struct rbtNode \*x = NULL, \*y = NULL, \*z;

z=root;

if((z->left ==NULL ) &&(z->right==NULL) && (z->key==var))

{ root=NULL;

printf("\nRBTREE is empty\n");

return;

}

while(z->key !=var && z!=NULL)

{ if(var<z->key)

z=z->left;

else

z=z->right;

if(z== NULL)

return;

}

if((z->left==NULL)||(z->right==NULL))

{ y = z;

}

else

{ y = successor(z);

}

if (y->left!=NULL)

{ x = y->left;

}

else

{ if(y->right !=NULL)

x = y->right;

}

if((x!=NULL) && (y->parent !=NULL))

x->parent = y->parent;

if ((y !=NULL) && (x!=NULL) && (y->parent==NULL))

{ root=x;

}

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

{ y->parent->left = x;

}

else

{ y->parent->right = x;

}

if (y != z)

{ z->key = y->key;

}

if ((y!=NULL) && (x!=NULL) && (y->color == 'b'))

{ color-delete(x);

}

return y;

}

int main(int argc, char\* argv[])

{ int choice,val,data,var,fl=0;

while(1)

{ printf("\nRed Black Tree Management - Enter your choice :1:Insert 2:Delete 3:Traversal 4:Exit\n");

scanf("%d",&choice);

switch(choice)

{ case 1:printf("Enter the integer you want to add : ");

scanf("%d",&val);

insert(val);

break;

case 2:printf("Enter the integer you want to delete : ");

scanf("%d",&var);

delete(var);

break;

case 3:traversal(root);

break;

case 4: fl=1;

break;

default:printf("\nInvalid Choice\n");

}

if(fl==1)

{ break;

}

}

return 0;

}