```
//Created By Ritwik Chandra Pandey
//On 25 Oct 2021
//Implementation of Red Black Tree: Deletion and Preorder Traversal


#include <stdio.h>
#include <stdlib.h>


typedef char COLOR;
struct node {
        int data;
    COLOR color;
        struct node *left, *right,*parent;
};

typedef struct node * RBNODE;
RBNODE root = NULL;

void leftRotate(RBNODE x) {
        RBNODE 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->data == x->parent->left->data)) {
                x->parent->left = y;
        }
        else {
                x->parent->right = y;
        }
        y->left = x;
        x->parent = y;
}
```

```
void rightRotate(RBNODE y) {
	RBNODE 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->data == y->parent->left->data)) {
		y->parent->left = x;
	}
	else {
		y->parent->right = x;
	}
	x->right = y;
	y->parent = x;
	return;
}

void colorInsert(RBNODE z) {
	RBNODE y=NULL;
	while ((z->parent != NULL) && (z->parent->color == 'r')) {
		if ( (z->parent->parent->left != NULL) && (z->parent->data == z->parent->parent->left->data)){

			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->data == z->parent->right->data)) {
```

```
                                        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->data == z->parent->left->data)) {
                                        z = z->parent;
                                        rightRotate(z);
                                }
                                z->parent->color = 'b';
                                z->parent->parent->color = 'r';
                                leftRotate(z->parent->parent);
                        }
                }

        }
        root->color = 'b';
}
int searchNodeInRB(int val) {
        RBNODE temp = root;
        int diff;
        while (temp != NULL) {
                if (val > temp->data) {
                        temp = temp->right;
                }
```

```
                else if (val < temp->data) {
                        temp = temp->left;
                }
                else
                {
                        return 1;
                }
        }
        return 0;
}
void insertNodeInRB(int ele) {
        RBNODE x,y;
        RBNODE z = (RBNODE)malloc(sizeof(struct node));
        z->data = ele;
        z->left = NULL;
        z->right = NULL;
        z->color = 'r';
        x=root;
        if(searchNodeInRB(ele)==1) {
                printf("Entered element already exists in the RBTree.\n");
                return;
        }
        if ( root == NULL ) {
                root = z;
                root->color = 'b';
                return;
        }
        while ( x != NULL) {
                y = x;
                if ( z->data < x->data) {
                        x = x->left;
                }
                else
                        x = x->right;
        }
        z->parent = y;
        if ( y == NULL) {
                root = z;
        }
        else if( z->data < y->data ) {
```

```
                y->left = z;
        }
        else
                y->right = z;
        colorInsert(z);
        return;
}
void preorderInRB(RBNODE root) {
        if(root!=NULL){
                printf("%d(%c)  ",root->data,root->color);
                preorderInRB(root->left);
                preorderInRB(root->right);
        }
}

RBNODE min(RBNODE x) {
        while(x->left!=NULL){
                x= x->left;
        }
        return x;
}

RBNODE successor(RBNODE x) {
        RBNODE y;
        if(x->right!=NULL){
        return  min(x->right);

        }
        y = x->parent;
        while(y!=NULL && y->right==x){
                x = y;
                x = y->parent;
        }
        return y;
}

void colorDelete(RBNODE x) {
        while(x!=root && x->color=='b'){
                RBNODE w;
                if(x->parent->left!=NULL && x->parent->left == x){
```

```
		w=x->parent->right;
		if(w!=NULL && w->color=='r'){
			w->color='b';
			x->parent->color = 'r';
			leftRotate(x->parent);
			x->parent->right= w;
		}
		if((w!=NULL && w->left!=NULL && w->right!=NULL) && (w->left->color == 'b' && w->right->color == 'b')){
			w->color='r';
			x = x->parent;
		}else if(w!=NULL && w->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->left->color=='b' && w->right->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';
}
void deleteNodeInRB(int ele) {
        RBNODE x =NULL, y = NULL, z;
        z = root;
        if(z->right==NULL && z->left == NULL  && z->data == ele){
                root = NULL;
                printf("Element %d deleted from RBTree.\n",z->data);
                return;


        }
        while(z->data!=ele && z!=NULL){
                if(ele<z->data){
                        z = z->left;
                }else{
                        z = z->right;
                }
                if(z==NULL){
                        printf("Element %d not found in RBTree.\n",ele);
                        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(x!=NULL && y!=NULL && y->parent==NULL){
			root = x;

		}
		else if(y->parent->left == y){
			y->parent->left = x;
		}
		else{
			y->parent->right = x;
		}
		if(y!=z){
			z->data = y->data;
		}
		if(x!=NULL && y!=NULL && y->color == 'b'){
			colorDelete(x);
		}
		printf("Element %d deleted from RBTree.\n", ele);
	}



void main() {
	int ele, op;
	while(1)
```

```
	{
		printf("1.Insert 2.Delete 3.Preorder Traverasal 4.Exit\n");
		printf("Enter your option : ");
		scanf("%d", &op);
		switch(op) {
			case 1:printf("Enter an element to be inserted : ");
				scanf("%d", &ele);
				insertNodeInRB(ele);
				break;
			case 2:printf("Enter an element to be deleted : ");
				scanf("%d", &ele);
				deleteNodeInRB(ele);
				break;
			case 3:
				if(root == NULL) {
					printf("RBTree is empty.\n");
				}
				else {
					printf("Elements of the RB tree (pre-order traversal): ");
					preorderInRB(root);
					printf("\n");
				}
				break;
			case 4:exit(0);
		}
	}
}
```