```
#include <stdio.h>
#include <stdlib.h>
int kk;

struct node //Node
{
        int key;
        struct node *parent;
        struct node *left;
        struct node *right;
        char color;
} * root, *null, *r;

void LeftRotate(struct node *t, struct node *x) //left rotate function
{
        struct node *y;
        y = x->right;
        x->right = y->left; //turn y's left subtree into x's right
subtree

        if (y->left != null)
        {
                y->left->parent = x;
        }
        y->parent = x->parent;

        if (x->parent == null)
                root = y;
        else if (x == x->parent->left)
                x->parent->left = y;
        else
                x->parent->right = y;

        y->left = x; //put x on y's left
        x->parent = y;
}

void RightRotate(struct node *t, struct node *y) //right rotate
function
{
        struct node *x;
        x = y->left;
        y->left = x->right; //turn x's right subtree into y's left
subtree

        if (x->right != null)
        {
                x->right->parent = y;
        }
        x->parent = y->parent;

        if (y->parent == null)
                root = x;
        else if (y == y->parent->right)
                y->parent->right = x;
        else
```

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

        x->right = y; //put y on x's left
        y->parent = x;
}

void Ifixup(struct node *a, struct node *z) //insert fixup function
{

        struct node *y = (struct node *)malloc(sizeof(struct node));
        while (z->parent->color == 'r')
        {
                if (z->parent == z->parent->parent->left)
                {
                        y = z->parent->parent->right;
                        if (y->color == 'r') //Case 1
                        {
                                z->parent->color = 'b';
                                y->color = 'b';
                                z->parent->parent->color = 'r';
                                z = z->parent->parent;
                        }
                        else if (z == z->parent->right) //Case 2
                        {
                                z = z->parent;
                                LeftRotate(root, z);
                        }
                        else //Case 3
                        {
                                z->parent->color = 'b';
                                z->parent->parent->color = 'r';
                                RightRotate(root, z->parent->parent);
                        }
                }

                else
                {
                        y = z->parent->parent->left;
                        if (y->color == 'r') //Case 1
                        {
                                z->parent->color = 'b';
                                y->color = 'b';
                                z->parent->parent->color = 'r';
                                z = z->parent->parent;
                        }
                        else if (z == z->parent->left) //Case 2
                        {
                                z = z->parent;
                                RightRotate(root, z);
                        }

                        else //Case 3
                        {
                                z->parent->color = 'b';
                                z->parent->parent->color = 'r';
                                LeftRotate(root, z->parent->parent);
                        }
```

```
                }
        }

        root->color = 'b';
}
void insert(struct node *tnode, int value)
{
        int count = 1;
        int k = 1;
        k = kk;
        struct node *x, *y, *z;
        z = (struct node *)malloc(sizeof(struct node)); //Allocating a
new node
        z->key = value;
        z->left = z->right = z->parent = null;
        z->color = 'r';

        y = null;
        x = root;

        while (x != null) //while loop for searching the location
where to insert element

        {
                y = x;

                if (z->key < x->key)
                        x = x->left;
                else
                        x = x->right;
        }

        z->parent = y;

        if (y == null)
        {

                z->color = 'b';
                root = z;
        }

        else if (z->key < y->key)
                y->left = z;

        else
                y->right = z;

        struct node *temp = (struct node *)malloc(sizeof(struct
node)); // Allocating a temporary node
        temp = z;

        while (temp->parent->color == 'r')
        {

                count++;
                temp = temp->parent;
        }
```

```
        if (count > k)
        {
                while (k - 1)
                {
                        z = z->parent;
                        k--;
                }

                Ifixup(root, z);
        }
}

struct node *search(struct node *a, int m)
{
        if (a == null)
                return null;
        else if (m = a->key)
                return a;

        else if (m < a->key)
        {
                search(a->left, m);
        }
        else if (m > a->key)
        {
                search(a->right, m);
        }
}

void RBtransplant(struct node *tree, struct node *u, struct node *v)
//Function for transplanting(interchange) nodes in the Tree

{
        if (u->parent == null)
                root = v;

        else if (u == u->parent->left)
                u->parent->left = v;

        else
                u->parent->right = v;
        v->parent = u->parent;
}

struct node *successor(struct node *s) // Function for finding the
immediate successor of a node in the tree

{
        while (s != null && s->left != null)
                s = s->left;
        return s;
}

void RB_Deletefix(struct node *d, struct node *x)

{
```

```
        while (x != root && x->color == 'b')
        {
                struct node *w = (struct node *)malloc(sizeof(struct
node));
                if (x == x->parent->left)
                {
                        w = x->parent->right;
                        if (w->color == 'r') //swapping color of w and
x parent
                        {
                                char temp;
                                temp = w->color;
                                w->color = x->parent->color;
                                x->parent->color = temp;
                                LeftRotate(root, x->parent);
                                w = x->parent->right;
                        }
                        /*in next line again i am checking the color
of w because in generic k  case we can again go to case 1 but for k=1
it is sure that after case 1 we will switch in 2,3 4 only*/
                        if (w->color == 'b' && w->left->color == 'b'
&& w->right->color == 'b')
                        {
                                w->color = 'r';
                                x = x->parent;
                        }

                        else if (w->color == 'b' && w->right->color ==
'b')
                        {
                                w->left->color = 'b';
                                w->color = 'r';
                                RightRotate(root, w);
                                w = x->parent->right;
                        }
                        else if (w->color == 'b' && w->right->color ==
'r')
                        {
                                w->color = x->parent->color;
                                x->parent->color = 'b';
                                w->right->color = 'b';
                                LeftRotate(root, x->parent);
                                x = root;
                        }
                }

                else
                {
                        w = x->parent->left;
                        if (w->color == 'r')
                        {
                                char temp;
                                temp = w->color;
                                w->color = x->parent->color;
                                x->parent->color = temp;
                                RightRotate(root, x->parent);
                                w = x->parent->left;
```

```
                        }

                        if (w->color == 'b' && w->left->color == 'b'
&& w->right->color == 'b')
                        {
                                w->color = 'r';
                                x = x->parent;
                        }

                        else if (w->color == 'b' && w->left->color ==
'b')
                        {
                                w->right->color = 'b';
                                w->color = 'r';
                                LeftRotate(root, w);
                                w = x->parent->left;
                        }
                        else if (w->color == 'b' && w->left->color ==
'r')
                        {
                                w->color = x->parent->color;
                                x->parent->color = 'b';
                                w->left->color = 'b';
                                RightRotate(root, x->parent);
                                x = root;
                        }
                }
        }
}

void rbDelete(struct node *d, int data) //Function too delete a node
from the Red-Black Tree
{
        char yo;

        struct node *z = (struct node *)malloc(sizeof(struct node));
        struct node *y = (struct node *)malloc(sizeof(struct node));
        struct node *x = (struct node *)malloc(sizeof(struct node));
        z = search(root, data);
        y = z;
        yo = y->color;

        if (z->left == null)
        {
                x = z->right;
                RBtransplant(root, z, z->right);
        }

        else if (z->right == null)
        {
                x = z->left;
                RBtransplant(root, z, z->left);
        }
        else
        {
                y = successor(z->right);
                yo = y->color;
```

```
                x = y->right;
                if (y->parent == z)
                {
                        x->parent = y;
                }
                else
                {
                        RBtransplant(root, y, y->right);
                        y->right = z->right;
                        y->right->parent = y;
                }
                RBtransplant(root, z, y);
                y->left = z->left;
                y->left->parent = y;
                y->color = z->color;
        }
        if (yo == 'b')
                RB_Deletefix(root, x);
}

void printInorder(struct node *node)
{

        if (node == null)
                return;
        printInorder(node->left);
        printf("%d,%c   ", node->key, node->color);
        printInorder(node->right);
}

void printPreorder(struct node *node)
{
        if (node == null)
                return;

        printf("%d,%c   ", node->key, node->color);
        printPreorder(node->left);
        printPreorder(node->right);
}

int main()
{

        int num, a;
        printf("Enter the value of k");
        scanf("%d", &kk);

        null = (struct node *)malloc(sizeof(struct node));
        null->parent = null;
        null->right = null;
        null->left = null;
        null->color = 'b';
        null->key = 0;
        root = null;
        printf("\nOPERATIONS ---");
        printf("\n1 - Insert an element into tree\n");
```

```
        printf("2 - Delete an element from the tree\n");

        while (1)

        {

                printf("\nEnter your choice 1 for insertion and 2 for
deletion : ");

                scanf("%d", &num);

                switch (num)

                {

                case 1:
                        printf("number of elements to be inserted");
                        scanf("%d", &a);
                        for (int i = 1; i <= a; i++)
                        {
                                int data;
                                printf("enter the key");
                                scanf("%d", &data);
                                insert(root, data);
                        }
                        printPreorder(root);
                        printf("\n");
                        printInorder(root);
                        break;
                case 2:
                        printf("number of elements to be deleted");
                        scanf("%d", &a);
                        for (int i = 1; i <= a; i++)
                        {
                                int data;
                                printf("enter the key");
                                scanf("%d", &data);
                                rbDelete(root, data);
                        }
                        printPreorder(root);
                        printf("\n");
                        printInorder(root);

                        break;
                }
        }

        return 0;
}
```