#include<stdio.h>

#include<stdlib.h>

struct Node

{

    int key;

    struct Node \*left;

    struct Node \*right;

    int height;

};

int max(int a, int b);

int height(struct Node \*N)

{

    if (N == NULL)

        return 0;

    return 1+max(height(N->left), height(N->right));

}

int max(int a, int b)

{

    return (a > b)? a : b;

}

struct Node\* newNode(int key)

{

    struct Node\* node = (struct Node\*)

                        malloc(sizeof(struct Node));

    node->key = key;

    node->left = NULL;

    node->right = NULL;

    node->height = 0;

    return(node);

}

#include <bits/stdc++.h>

using namespace std;

struct AVLwithparent {

    struct AVLwithparent\* left;

    struct AVLwithparent\* right;

    int key;

    struct AVLwithparent\* par;

    int height;

};

void Updateheight(struct AVLwithparent\* root)

{

    if (root != NULL) {

        int val = 1;

        if (root->left != NULL)

            val = root->left->height + 1;

        if (root->right != NULL)

            val = max(

                val, root->right->height + 1);

        root->height = val;

    }

}

struct AVLwithparent\* LLR(

    struct AVLwithparent\* root)

{

    struct AVLwithparent\* tmpnode = root->left;

    root->left = tmpnode->right;

    if (tmpnode->right != NULL)

        tmpnode->right->par = root;

    tmpnode->right = root;

    tmpnode->par = root->par;

    root->par = tmpnode;

    if (tmpnode->par != NULL

        && root->key < tmpnode->par->key) {

        tmpnode->par->left = tmpnode;

    }

    else {

        if (tmpnode->par != NULL)

            tmpnode->par->right = tmpnode;

    }

    root = tmpnode;

    Updateheight(root->left);

    Updateheight(root->right);

    Updateheight(root);

    Updateheight(root->par);

    return root;

}

struct AVLwithparent\* RRR(

    struct AVLwithparent\* root)

{

    struct AVLwithparent\* tmpnode = root->right;

    root->right = tmpnode->left;

    if (tmpnode->left != NULL)

        tmpnode->left->par = root;

    tmpnode->left = root;

    tmpnode->par = root->par;

    root->par = tmpnode;

    if (tmpnode->par != NULL

        && root->key < tmpnode->par->key) {

        tmpnode->par->left = tmpnode;

    }

    else {

        if (tmpnode->par != NULL)

            tmpnode->par->right = tmpnode;

    }

    root = tmpnode;

    Updateheight(root->left);

    Updateheight(root->right);

    Updateheight(root);

    Updateheight(root->par);

    return root;

}

struct AVLwithparent\* LRR(

    struct AVLwithparent\* root)

{

    root->left = RRR(root->left);

    return LLR(root);

}

struct AVLwithparent\* RLR(

    struct AVLwithparent\* root)

{

    root->right = LLR(root->right);

    return RRR(root);

}

struct AVLwithparent\* Insert(

    struct AVLwithparent\* root,

    struct AVLwithparent\* parent,

    int key)

{

    if (root == NULL) {

        root = new struct AVLwithparent;

        if (root == NULL) {

            cout << "Error in memory" << endl;

        }

        else {

            root->height = 1;

            root->left = NULL;

            root->right = NULL;

            root->par = parent;

            root->key = key;

        }

    }

    else if (root->key > key) {

        root->left = Insert(root->left,

                            root, key);

        int firstheight = 0;

        int secondheight = 0;

        if (root->left != NULL)

            firstheight = root->left->height;

        if (root->right != NULL)

            secondheight = root->right->height;

        if (abs(firstheight

                - secondheight)

            == 2) {

            if (root->left != NULL

                && key < root->left->key) {

                root = LLR(root);

            }

            else {

                root = LRR(root);

            }

        }

    }

    else if (root->key < key) {

        root->right = Insert(root->right, root, key);

        int firstheight = 0;

        int secondheight = 0;

        if (root->left != NULL)

            firstheight = root->left->height;

        if (root->right != NULL)

            secondheight = root->right->height;

        if (abs(firstheight

                - secondheight)

            == 2) {

            if (root->right != NULL

                && key < root->right->key) {

                root = RLR(root);

            }

            else {

                root = RRR(root);

            }

        }

    }

    else {

    }

    Updateheight(root);

    return root;

}

bool AVLsearch(

    struct AVLwithparent\* root, int key)

{

    if (root == NULL)

        return false;

    else if (root->key == key)

        return true;

    else if (root->key > key) {

        bool val = AVLsearch(root->left, key);

        return val;

    }

    else {

        bool val = AVLsearch(root->right, key);

        return val;

    }

}

int main()

{

    struct AVLwithparent\* root;

    root = NULL;

    root = Insert(root, NULL, 10);

    root = Insert(root, NULL, 20);

    root = Insert(root, NULL, 30);

    root = Insert(root, NULL, 40);

    root = Insert(root, NULL, 50);

    root = Insert(root, NULL, 25);

    bool found = AVLsearch(root, 40);

    if (found)

        cout << "value found";

    else

        cout << "value not found";

    return 0;

}

#include<stdio.h>

#include<stdlib.h>

struct Node

{

    int key;

    struct Node \*left;

    struct Node \*right;

    int height;

};

int max(int a, int b);

int height(struct Node \*N)

{

    if (N == NULL)

        return 0;

    return N->height;

}

int max(int a, int b)

{

    return (a > b)? a : b;

}

struct Node\* newNode(int key)

{

    struct Node\* node = (struct Node\*)

                        malloc(sizeof(struct Node));

    node->key = key;

    node->left = NULL;

    node->right = NULL;

    node->height = 1;

    return(node);

}

struct Node \*rightRotate(struct Node \*y)

{

    struct Node \*x = y->left;

    struct Node \*T2 = x->right;

    x->right = y;

    y->left = T2;

    y->height = max(height(y->left), height(y->right))+1;

    x->height = max(height(x->left), height(x->right))+1;

    return x;

}

struct Node \*leftRotate(struct Node \*x)

{

    struct Node \*y = x->right;

    struct Node \*T2 = y->left;

    y->left = x;

    x->right = T2;

    x->height = max(height(x->left), height(x->right))+1;

    y->height = max(height(y->left), height(y->right))+1;

    return y;

}

int getBalance(struct Node \*N)

{

    if (N == NULL)

        return 0;

    return height(N->left) - height(N->right);

}

struct Node\* insert(struct Node\* node, int key)

{

    if (node == NULL)

        return(newNode(key));

    if (key < node->key)

        node->left = insert(node->left, key);

    else if (key > node->key)

        node->right = insert(node->right, key);

    else

        return node;

    node->height = 1 + max(height(node->left),

                        height(node->right));

    int balance = getBalance(node);

    if (balance > 1 && key < node->left->key)

        return rightRotate(node);

    if (balance < -1 && key > node->right->key)

        return leftRotate(node);

    if (balance > 1 && key > node->left->key)

    {

        node->left = leftRotate(node->left);

        return rightRotate(node);

    }

    if (balance < -1 && key < node->right->key)

    {

        node->right = rightRotate(node->right);

        return leftRotate(node);

    }

    return node;

}

struct Node \* minValueNode(struct Node\* node)

{

    struct Node\* current = node;

    while (current->left != NULL)

        current = current->left;

    return current;

}

struct Node\* deleteNode(struct Node\* root, int key)

{

    if (root == NULL)

        return root;

    if ( key < root->key )

        root->left = deleteNode(root->left, key);

    else if( key > root->key )

        root->right = deleteNode(root->right, key);

    else

    {

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

        {

            struct Node \*temp = root->left ? root->left :

                                            root->right;

            if (temp == NULL)

            {

                temp = root;

                root = NULL;

            }

            else

            \*root = \*temp;

            free(temp);

        }

        else

        {

            struct Node\* temp = minValueNode(root->right);

            root->key = temp->key;

            root->right = deleteNode(root->right, temp->key);

        }

    }

    if (root == NULL)

    return root;

    root->height = 1 + max(height(root->left),

                        height(root->right));

    int balance = getBalance(root);

    if (balance > 1 && getBalance(root->left) >= 0)

        return rightRotate(root);

    if (balance > 1 && getBalance(root->left) < 0)

    {

        root->left = leftRotate(root->left);

        return rightRotate(root);

    }

    if (balance < -1 && getBalance(root->right) <= 0)

        return leftRotate(root);

    if (balance < -1 && getBalance(root->right) > 0)

    {

        root->right = rightRotate(root->right);

        return leftRotate(root);

    }

    return root;

}

void preOrder(struct Node \*root)

{

    if(root != NULL)

    {

        printf("%d ", root->key);

        preOrder(root->left);

        preOrder(root->right);

    }

}

int main()

{

struct Node \*root = NULL;

    root = insert(root, 9);

    root = insert(root, 5);

    root = insert(root, 10);

    root = insert(root, 0);

    root = insert(root, 6);

    root = insert(root, 11);

    root = insert(root, -1);

    root = insert(root, 1);

    root = insert(root, 2);

    printf("Preorder traversal of the constructed AVL "

        "tree is \n");

    preOrder(root);

    root = deleteNode(root, 10);

    printf("\nPreorder traversal after deletion of 10 \n");

    preOrder(root);

    return 0;

}

