#include <stdio.h>

#include <stdlib.h>

*// Define the structure of an AVL Tree node*

struct Node {

    int key;

    struct Node \*left;

    struct Node \*right;

    int height;

};

*// Function to get the height of the tree*

int height(struct Node \**node*) {

    if (*node* == NULL)

        return 0;

    return *node*->height;

}

*// Function to get the max of two integers*

int max(int *a*, int *b*) {

    return (*a* > *b*) ? *a* : *b*;

}

*// Create a new node*

struct Node\* createNode(int *key*) {

    struct Node\* node = (struct Node\*) malloc(sizeof(struct Node));

    node->key = *key*;

    node->left = NULL;

    node->right = NULL;

    node->height = 1; *// New node is initially at leaf (height 1)*

    return node;

}

*// Right rotate (LL rotation)*

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

    struct Node \*x = *y*->left;

    struct Node \*T2 = x->right;

*// Perform rotation*

    x->right = *y*;

*y*->left = T2;

*// Update heights*

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

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

    return x; *// New root*

}

*// Left rotate (RR rotation)*

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

    struct Node \*y = *x*->right;

    struct Node \*T2 = y->left;

*// Perform rotation*

    y->left = *x*;

*x*->right = T2;

*// Update heights*

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

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

    return y; *// New root*

}

*// Get Balance Factor*

int getBalance(struct Node \**node*) {

    if (*node* == NULL)

        return 0;

    return height(*node*->left) - height(*node*->right);

}

*// Recursive function to insert a key in the subtree and balance it*

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

*// 1. Perform the normal BST insertion*

    if (*node* == NULL)

        return createNode(*key*);

    if (*key* < *node*->key)

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

    else if (*key* > *node*->key)

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

    else *// Duplicate keys not allowed*

        return *node*;

*// 2. Update height*

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

*// 3. Get balance factor*

    int balance = getBalance(*node*);

*// 4. Balance the tree*

*// LL Case*

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

        return rightRotate(*node*);

*// RR Case*

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

        return leftRotate(*node*);

*// LR Case*

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

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

        return rightRotate(*node*);

    }

*// RL Case*

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

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

        return leftRotate(*node*);

    }

*// Return unchanged node*

    return *node*;

}

*// Inorder traversal*

void inorder(struct Node\* *root*) {

    if (*root* != NULL) {

        inorder(*root*->left);

        printf("%d ", *root*->key);

        inorder(*root*->right);

    }

}

*// Driver code*

int main() {

    struct Node \*root = NULL;

*// Sample insertions*

    root = insert(root, 30);

    root = insert(root, 20);

    root = insert(root, 10);

    root = insert(root, 25);

    root = insert(root, 40);

    root = insert(root, 50);

    root = insert(root, 5);

    printf("Inorder traversal of AVL Tree:\n");

    inorder(root);

    return 0;

}