NAME: Venkateswar L
BRANCH: Computer Science and Engineering SEC: F
ROLL NO.: 230701376

PROGRAM: Implementation Of AVL Tree

Write a function in C program to insert a new node with a given value into an AVL tree. Ensure that the tree remains balanced after insertion by performing rotations if necessary. Repeat the above operation to delete a node from AVL tree.

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

typedef struct Node {
    int key;
    struct Node* left;
    struct Node* right;
    int height;
} Node;

int height(Node* node) {
    if (node == NULL)
        return 0;
    return node->height;
}

int max(int a, int b) {
    return (a > b) ? a : b;
}

Node* newNode(int key) {
```

```
    Node* node = (Node*)malloc(sizeof(Node));
    node->key = key;
    node->left = NULL;
    node->right = NULL;
    node->height = 1;
    return node;
}

Node* rightRotate(Node* y) {
    Node* x = y->left;
    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;
}

Node* leftRotate(Node* x) {
    Node* y = x->right;
    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(Node* N) {
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}



Node* insert(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;
}

Node* deleteNode(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)) {
            Node* temp = root->left ? root->left : root->right;
```

```
        if (temp == NULL) {
            temp = root;
            root = NULL;
        } else
            *root = *temp;

        free(temp);
    } else {

        Node* temp = root->right;
        while (temp->left != NULL)
            temp = temp->left;


        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(Node* root) {
    if (root != NULL) {
        printf("%d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}
```

NAME: Venkateswar L
BRANCH: Computer Science and Engineering SEC: F
ROLL NO.: 230701376

```
int main() {
    Node* root = NULL;
    int key;
    int n,value;
    printf("Enter number of nodes to be inserted:");
    scanf("%d",&n);
    for (int i=0;i<n;i++){
        printf("Enter data: ");
        scanf("%d",&value);
        root=insert(root,value);
    }

    printf("Preorder traversal of the AVL tree after insertion: ");
    preOrder(root);
    printf("\n");



    printf("enter key to delete: ");
    scanf("%d",&key);
    root = deleteNode(root,key);

    printf("Preorder traversal of the AVL tree after deletion of node with key %d:
",key);
    preOrder(root);
    printf("\n");
```

```
    return 0;
}
```