1. WRITE A C PROGRAM FOR AVL(INSERT,DELETE,SEARCH).

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int key;
    struct Node* left;
    struct Node* right;
    int height;
} Node;
int height(Node* N) {
    if (N == NULL) return 0;
    return N->height;
}
int max(int a, int b) {
    return (a > b) ? a : b;
}
Node* createNode(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 createNode(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 = max(height(node->left), height(node->right)) + 1;
    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* minValueNode(Node* node) {
    Node* current = node;
    while (current->left != NULL)
        current = current->left;
    return current;
}
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 = minValueNode(root->right);
            root->key = temp->key;
            root->right = deleteNode(root->right, temp->key);
        }
    }
    if (root == NULL) return root;
    root->height = max(height(root->left), height(root->right)) + 1;
    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;
}
Node* search(Node* root, int key) {
    if (root == NULL || root->key == key)
        return root;

    if (root->key < key)
        return search(root->right, key);

    return search(root->left, key);
}
void inorder(Node* root) {
```

```
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->key);
        inorder(root->right);
    }
}
int main() {
    Node* root = NULL;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 40);
    root = insert(root, 50);
    root = insert(root, 25);
    printf("In-order traversal of the AVL tree:\n");
    inorder(root);
    printf("\n");
    int key = 30;
    Node* result = search(root, key);
    if (result != NULL) {
        printf("Node %d found in the AVL tree.\n", key);
    } else {
        printf("Node %d not found in the AVL tree.\n", key);
    }
    root = deleteNode(root, 20);
    printf("In-order traversal after deletion:\n");
    inorder(root);
    printf("\n");
    return 0;
}
```

SAMPLE OUTPUT:

```
In-order traversal of the AVL tree:
10 20 25 30 40 50
Node 30 found in the AVL tree.
In-order traversal after deletion:
10 25 30 40 50
```