AVL Tree

Write a C Program on Insert a node in AVL tree and perform the operations like insert,delete,search on following :

i)18,8,18,5,11,17,4

ii)25,20,36,10,22,30,40,12,28,38,48

iii)1,2,3,4,5,6,7,8

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct AVLNode {
   int key;
   struct AVLNode* left;
   struct AVLNode* right;
   int height;
};
struct AVLNode* createNode(int key) {
   struct AVLNode* node = (struct AVLNode*)malloc(sizeof(struct AVLNode));
   node->key = key;
   node->left = NULL;
   node->right = NULL;
   node->height = 1;
   return node;
}
int height(struct AVLNode* node) {
   if (node == NULL)
      return 0;
   return node->height;
}
int getBalance(struct AVLNode* node) {
   if (node == NULL)
      return 0;
```

```
    return height(node->left) - height(node->right);
}
struct AVLNode* rightRotate(struct AVLNode* y) {
    struct AVLNode* x = y->left;
    struct AVLNode* T2 = x->right;
    x->right = y;
    y->left = T2;
    y->height = 1 + (height(y->left) > height(y->right) ? height(y->left) : height(y->right));
    x->height = 1 + (height(x->left) > height(x->right) ? height(x->left) : height(x->right));
    return x;
}
struct AVLNode* leftRotate(struct AVLNode* x) {
    struct AVLNode* y = x->right;
    struct AVLNode* T2 = y->left;
    y->left = x;
    x->right = T2;
    x->height = 1 + (height(x->left) > height(x->right) ? height(x->left) : height(x->right));
    y->height = 1 + (height(y->left) > height(y->right) ? height(y->left) : height(y->right));
    return y;
}
struct AVLNode* insert(struct AVLNode* 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 = 1 + (height(node->left) > height(node->right) ? 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 AVLNode* minValueNode(struct AVLNode* node) {
    struct AVLNode* current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}
struct AVLNode* deleteNode(struct AVLNode* 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 AVLNode* temp = root->left ? root->left : root->right;
            if (temp == NULL) {
                temp = root;
                root = NULL;
            } else
                *root = *temp;
            free(temp);
        } else {
            struct AVLNode* temp = minValueNode(root->right);
            root->key = temp->key;
            root->right = deleteNode(root->right, temp->key);
```

```
        }
    }
    if (root == NULL)
        return root;
    root->height = 1 + (height(root->left) > height(root->right) ? 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;
}
struct AVLNode* search(struct AVLNode* 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(struct AVLNode* root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->key);
        inorder(root->right);
    }
}
int main() {
```

```
struct AVLNode* root1 = NULL;
struct AVLNode* root2 = NULL;
struct AVLNode* root3 = NULL;
int values1[] = {18, 8, 18, 5, 11, 17, 4};
int n1 = sizeof(values1) / sizeof(values1[0]);
for (int i = 0; i < n1; i++) {
   root1 = insert(root1, values1[i]);
}
printf(" after insertion: ");
inorder(root1);
printf("\n");
int searchKey1 = 11;
struct AVLNode* foundNode1 = search(root1, searchKey1);
if (foundNode1 != NULL) {
   printf("Key %d found in the first AVL tree.\n", searchKey1);
} else {
   printf("Key %d not found in the first AVL tree.\n", searchKey1);
}
int deleteKey1 = 8;
root1 = deleteNode(root1, deleteKey1);
printf(" after deleting %d: ", deleteKey1);
inorder(root1);
printf("\n");
int values2[] = {25, 20, 36, 10, 22, 30, 40, 12, 28, 38, 48};
int n2 = sizeof(values2) / sizeof(values2[0]);
for (int i = 0; i < n2; i++) {
   root2 = insert(root2, values2[i]);
}
printf(" after insertion: ");
inorder(root2);
printf("\n");
int searchKey2 = 30;
struct AVLNode* foundNode2 = search(root2, searchKey2);
if (foundNode2 != NULL) {
   printf("Key %d found in the second AVL tree.\n", searchKey2);
} else {
```

```
        printf("Key %d not found in the second AVL tree.\n", searchKey2);
    }
    int deleteKey2 = 36;
    root2 = deleteNode(root2, deleteKey2);
    printf("e after deleting %d: ", deleteKey2);
    inorder(root2);
    printf("\n");
    int values3[] = {1, 2, 3, 4, 5, 6, 7, 8};
    int n3 = sizeof(values3) / sizeof(values3[0]);
    for (int i = 0; i < n3; i++) {
        root3 = insert(root3, values3[i]);
    }
    printf(" after insertion: ");
    inorder(root3);
    printf("\n");
    int searchKey3 = 5;
    struct AVLNode* foundNode3 = search(root3, searchKey3);
    if (foundNode3 != NULL) {
        printf("Key %d found in the third AVL tree.\n", searchKey3);
    } else {
        printf("Key %d not found in the third AVL tree.\n", searchKey3);
    }
    int deleteKey3 = 4;
    root3 = deleteNode(root3, deleteKey3);
    printf(" after deleting %d: ", deleteKey3);
    inorder(root3);
    printf("\n");
    return 0;
}
```

OUTPUT:

```
after insertion: 4 5 8 11 17 18
Key 11 found in the first AVL tree.
 after deleting 8: 4 5 11 17 18
 after insertion: 10 12 20 22 25 28 30 36 38 40 48
Key 30 found in the second AVL tree.
```

e after deleting 36: 10 12 20 22 25 28 30 38 40 48

after insertion: 1 2 3 4 5 6 7 8

Key 5 found in the third AVL tree.

after deleting 4: 1 2 3 5 6 7 8