Write the c program in AVL for Insertion, Search and Deletion

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int key;
   struct Node *left, *right;
   int height;
};
int height(struct Node *N) {
   return (N == NULL) ? 0 : 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 = 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) {
   return (N == NULL) ? 0 : 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 = 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;
}
struct Node* search(struct 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 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, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 40);
    root = insert(root, 50);
    root = insert(root, 25);
    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);
    struct Node* foundNode = search(root, 20);
    if (foundNode != NULL)
        printf("\nNode with key 20 found\n");
    else
        printf("\nNode with key 20 not found\n");
    return 0;
}
```

Output:

Preorder traversal of the constructed AVL tree is:

30 20 10 25 40 50

Preorder traversal after deletion of 10**:**

30 20 25 40 50

Node with key 20 found