19.
```
#include <stdio.h>
#include <stdlib.h>

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

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

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

struct Node* createNode(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* node) {
    if (node == NULL) return 0;
    return height(node->left) - height(node->right);
```

```
}

struct Node* insert(struct 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; // no duplicates

    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* search(struct Node* root, int key) {
    if (root == NULL || root->key == key)
        return root;
    if (key < root->key)
        return search(root->left, key);
    return search(root->right, 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);

    printf("AVL tree preorder: ");
    preorder(root);
    printf("\n");

    int key = 20;
    if (search(root, key) != NULL)
        printf("Key %d found.\n", key);
    else
        printf("Key %d not found.\n", key);

    return 0;
}
```

Output

```
AVL tree preorder: 20 10 30
Key 20 found.


=== Code Execution Successful ===
```