A.Insertion of AVL tree

```
AVLNode* insertNode(AVLNode* root, int data) {
    // Perform standard BST insertion
    if (root == NULL) {
        AVLNode* newNode = (AVLNode*)malloc(sizeof(AVLNode));
        newNode->data = data;
        newNode->left = newNode->right = NULL;
        newNode->height = 1;
        return newNode;
    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else if (data > root->data) {
        root->right = insertNode(root->right, data);
    } else {
        return root;
    }
    // Update height of the current node
    root->height = 1 + max(getHeight(root->left), getHeight(root->right));
    int balance = getBalance(root);
    if (balance > 1 && data < root->left->data) {
        return rightRotate(root);
    }
    if (balance < -1 && data > root->right->data) {
        return leftRotate(root);
    }
    if (balance > 1 && data > root->left->data) {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }
    if (balance < -1 && data < root->right->data) {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }
    return root;
}
```

B. Delation operation for AVL tree

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int key;
```

```
    struct Node *left;
    struct Node *right;
    int height;
};
int getHeight(struct Node *node) {
    if (node == NULL)
        return 0;
    return node->height;
}
void updateHeight(struct Node *node) {
    node->height = 1 + max(getHeight(node->left), getHeight(node->right));
}
struct Node *rightRotate(struct Node *y) {
    struct Node *x = y->left;
    struct Node *T = x->right;
    x->right = y;
    y->left = T;
    updateHeight(y);
    updateHeight(x);
    return x;
}
struct Node *leftRotate(struct Node *x) {
    struct Node *y = x->right;
    struct Node *T = y->left;
    y->left = x;
    x->right = T;
    updateHeight(x);
    updateHeight(y);

    return y;
}
int getBalance(struct Node *node) {
    if (node == NULL)
        return 0;
    return getHeight(node->left) - getHeight(node->right);
}
struct Node *deleteNode(struct Node *root, int key) {
}
int main() {
    struct Node *root = NULL;
    return 0;
}
```

C.Search operation for AVL tree

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

```c
#include <stdlib.h>
// AVL Node structure
struct Node {
    int key;
    struct Node *left;
    struct Node *right;
    int height;
};
// Function to search for a key in an AVL tree
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);
    }
    else {
        return search(root->right, key);
    }
}
// Main function to demonstrate AVL node search
int main() {
    struct Node* root = NULL; // Initialize an empty AVL tree
    // Perform search operation
    int key_to_search = 10;
    struct Node* result = search(root, key_to_search);
    if (result != NULL) {
        printf("Node with key %d found in the AVL tree.", key_to_search);
    } else {
        printf("Node with key %d not found in the AVL tree.", key_to_search);
    }
    return 0;
}
```