What is an AVL Tree?

An **AVL Tree** is a type of **self-balancing Binary Search Tree (BST)** where the difference between the heights of the left and right subtrees is at most **1**.

Each node maintains a **balance factor**:

- `balance = height(left) - height(right)`
- Must be -1, 0, or +1.

If it's outside this range after insertion/deletion, we **rotate** to fix it.

Steps to Implement:

1. Define the Node structure.
2. Write helper functions:
    - `height()`
    - `max()`
    - `getBalance()`
3. Write rotation functions:
    - Left Rotation
    - Right Rotation
    - Left-Right and Right-Left
4. Insertion with balancing.

1. Node Structure

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

Each **Node** has:

- `key`: the value stored in the node.
- `left`, `right`: pointers to left and right children.
- `height`: height of the node, used to check balance.

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

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

// Utility: Get max of two integers
int max(int a, int b) {
    return (a > b) ? a : b;
}

// Utility: Get height of the tree
int height(struct Node *N) {
    if (N == NULL)
        return 0;
    return N->height;
}
```

```
// Allocate new node
struct Node* newNode(int key) {
    struct Node* node = (struct Node*) malloc(sizeof(struct Node));
    node->key = key;
    node->left = NULL;
    node->right = NULL;
    node->height = 1;  // New node is initially at leaf
    return node;
}

// Get balance factor of a node
int getBalance(struct Node *N) {
    if (N == NULL)
        return 0;
    return height(N->left) - height(N->right);
}

// Right rotate
struct Node* rightRotate(struct Node *y) {
    struct Node *x = y->left;
    struct Node *T2 = x->right;

    // Perform rotation
```

```
    x->right = y;
    y->left = T2;

    // Update heights
    y->height = max(height(y->left), height(y->right)) + 1;
    x->height = max(height(x->left), height(x->right)) + 1;

    // Return new root
    return x;
}

// Left rotate
struct Node* leftRotate(struct Node *x) {
    struct Node *y = x->right;
    struct Node *T2 = y->left;

    // Perform rotation
    y->left = x;
    x->right = T2;

    // Update heights
    x->height = max(height(x->left), height(x->right)) + 1;
    y->height = max(height(y->left), height(y->right)) + 1;
```

```
    // Return new root
    return y;
}

// Insert function
struct Node* insert(struct Node* node, int key) {
    // 1. Perform normal BST insertion
    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  // Duplicate keys not allowed
        return node;

    // 2. Update height
    node->height = 1 + max(height(node->left), height(node->right));

    // 3. Get balance factor
    int balance = getBalance(node);
```

```
// 4. Balance the tree
// Left Left Case
if (balance > 1 && key < node->left->key)
    return rightRotate(node);

// Right Right Case
if (balance < -1 && key > node->right->key)
    return leftRotate(node);

// Left Right Case
if (balance > 1 && key > node->left->key) {
    node->left = leftRotate(node->left);
    return rightRotate(node);
}

// Right Left Case
if (balance < -1 && key < node->right->key) {
    node->right = rightRotate(node->right);
    return leftRotate(node);
}

// return the unchanged node pointer
```

```
    return node;
}

// Inorder traversal
void inorder(struct Node *root) {
    if (root != NULL) {
        inorder(root->left);
        printf("%d ", root->key);
        inorder(root->right);
    }
}

// Driver code
int main() {
    struct Node *root = NULL;

    // Let's insert elements into the AVL tree
    int keys[] = {10, 20, 30, 40, 50, 25};
    for (int i = 0; i < 6; i++) {
        root = insert(root, keys[i]);
    }

    printf("Inorder traversal of the AVL tree is:\n");
```

```
    inorder(root);
    return 0;
}
```