AVL Tree Insertion Code

```
#include <iostream>
using namespace std;

class Node {
public:
    int data, height;
    Node *left, *right;

    Node(int value) {
        data = value;
        height = 1;
        left = right = nullptr;
    }
};

// Utility to get height
int getHeight(Node *root) {
    if (!root) return 0;
    return root->height;
}

// Utility to get balance factor
int getBalance(Node *root) {
    if (!root) return 0;
    return getHeight(root->left) - getHeight(root->right);
}

// Left Rotation
Node* leftRotation(Node *x) {
    Node *y = x->right;
    Node *T2 = y->left;

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

    // Update heights
```

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

    return y; // new root
}

// Right Rotation
Node* rightRotation(Node *y) {
    Node *x = y->left;
    Node *T2 = x->right;

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

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

    return x; // new root
}

// Insertion with balancing
Node* insert(Node *root, int key) {
    // Step 1: Normal BST insertion
    if (!root)
        return new Node(key);

    if (key < root->data)
        root->left = insert(root->left, key);
    else if (key > root->data)
        root->right = insert(root->right, key);
    else
        return root; // Duplicates not allowed

    // Step 2: Update height
    root->height = 1 + max(getHeight(root->left), getHeight(root->right));

    // Step 3: Get balance factor
    int balance = getBalance(root);
```

```
    // Step 4: Check for unbalanced cases and apply rotations

    // Left Left Case
    if (balance > 1 && key < root->left->data)
        return rightRotation(root);

    // Right Right Case
    if (balance < -1 && key > root->right->data)
        return leftRotation(root);

    // Left Right Case
    if (balance > 1 && key > root->left->data) {
        root->left = leftRotation(root->left);
        return rightRotation(root);
    }

    // Right Left Case
    if (balance < -1 && key < root->right->data) {
        root->right = rightRotation(root->right);
        return leftRotation(root);
    }

    // Return unchanged root if balanced
    return root;
}

// In-order traversal to verify result
void inorder(Node* root) {
    if (root) {
        inorder(root->left);
        cout << root->data << " ";
        inorder(root->right);
    }
}

int main() {
    Node *root = nullptr;

    root = insert(root, 20);
```

```
    root = insert(root, 30);
    root = insert(root, 40);
    root = insert(root, 60);
    root = insert(root, 80);
    root = insert(root, 10);
    root = insert(root, 100);
    root = insert(root, 95); // This will trigger RL rotation

    cout << "Inorder traversal of AVL tree: ";
    inorder(root);
    cout << endl;

    return 0;
}
```