```cpp
//AVL Tree
#include <iostream>

using namespace std;

struct AVLNode {
    int key;
    AVLNode* left, * right;
    int height;
};

int getHeight(AVLNode* root) {
    if (!root) return 0;
    return root->height;
}

void fixHeight(AVLNode*& root) {
    root->height = 1 + std::max(getHeight(root->left), getHeight(root->right));
}

void leftRotate(AVLNode*& root) {
    AVLNode* B = root->right;
    AVLNode* Y = B->left;

    B->left = root;
    root->right = Y;

    fixHeight(root);
    fixHeight(B);

    root = B;
}

void rightRotate(AVLNode*& root) {
    AVLNode* B = root->left;
    AVLNode* Y = B->right;

    B->right = root;
    root->left = Y;

    fixHeight(root);
    fixHeight(B);

    root = B;
}

int getBalanceFactor(AVLNode* node) {
    return getHeight(node->left) - getHeight(node->right);
```

```cpp
}

void insertAVLNode(AVLNode*& root, int key) {
    if (!root) {
        root = new AVLNode{ key, nullptr, nullptr, 1 };
        return;
    }

    if (key < root->key) {
        insertAVLNode(root->left, key);
    }
    else if (key > root->key) {
        insertAVLNode(root->right, key);
    }
    else return;

    fixHeight(root);

    int bf = getBalanceFactor(root);

    if (bf > 1 && key < root->left->key) {
        rightRotate(root);
        return;
    }

    if (bf < -1 && key > root->right->key) {
        leftRotate(root);
        return;
    }

    if (bf > 1 && key > root->left->key) {
        leftRotate(root->left);
        rightRotate(root);
        return;
    }

    if (bf < -1 && key < root->right->key) {
        rightRotate(root->right);
        leftRotate(root);
        return;
    }
}

AVLNode* findPredecessor(AVLNode* node) {
    if (!node->left) {
        std::cout << "This node does not have predecessor!";
        return nullptr;
    }

    AVLNode* y = node->left;
    while (y->right) y = y->right;
    return y;
}

void deleteAVLNode(AVLNode*& root, int key) {
    if (!root) return;
    else if (key < root->key) deleteAVLNode(root->left, key);
    else if (key > root->key) deleteAVLNode(root->right, key);
    else {
        if (!root->left) {
            AVLNode* temp = root->right;
            delete root;
            root = temp;
            return;
        }
        else if (!root->right) {
            AVLNode* temp = root->left;
            delete root;
            root = temp;
            return;
        }
        else {
            AVLNode* pred = findPredecessor(root);
            root->key = pred->key;

            deleteAVLNode(root->left, pred->key);
        }
    }

    fixHeight(root);

    int bf = getBalanceFactor(root);

    if (bf > 1 && getBalanceFactor(root->left) >= 0) {
        rightRotate(root);
    }

    else if (bf < -1 && getBalanceFactor(root->right) <= 0) {
        leftRotate(root);
    }

    else if (bf > 1 && getBalanceFactor(root->left) < 0) {
        leftRotate(root->left);
        rightRotate(root);
    }

    else if (bf < -1 && getBalanceFactor(root->right) > 0) {
        rightRotate(root->right);
        leftRotate(root);
    }
}
```