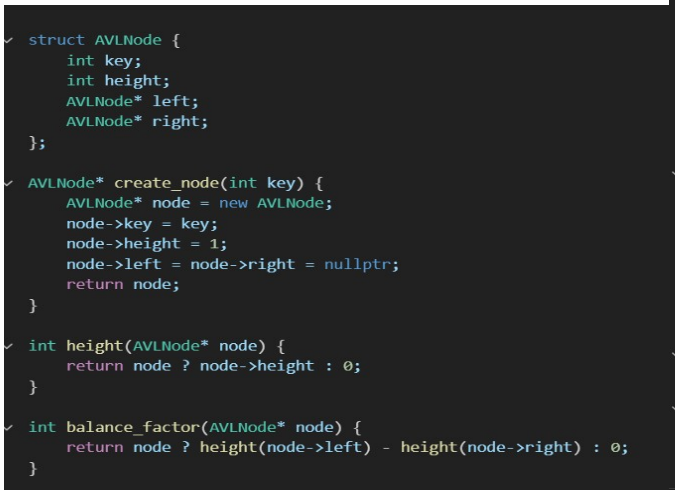
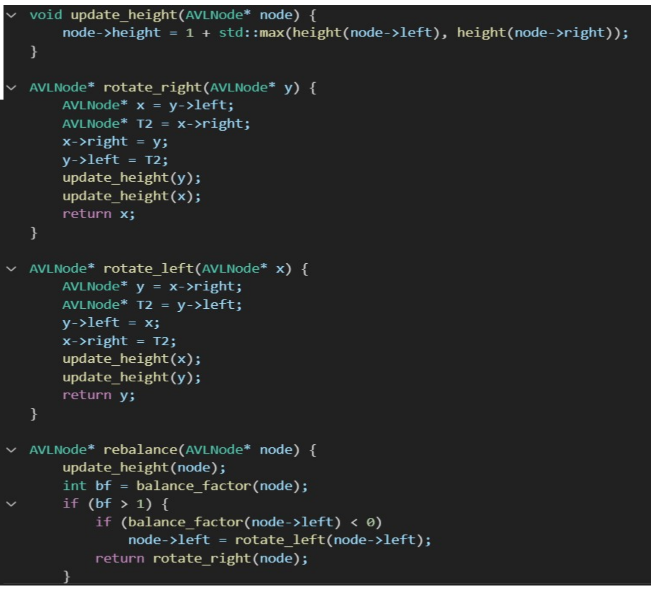
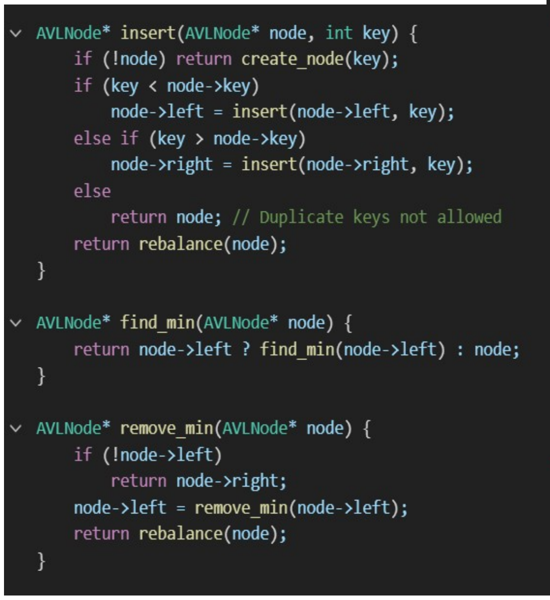


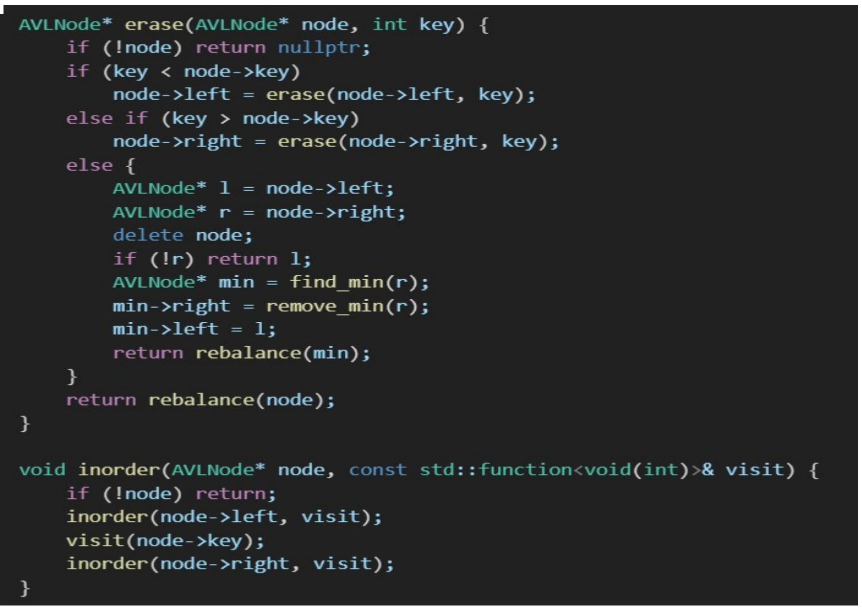
```cpp
struct AVLNode {
    int key;
    int height;
    AVLNode* left;
    AVLNode* right;
};

AVLNode* create_node(int key) {
    AVLNode* node = new AVLNode;
    node->key = key;
    node->height = 1;
    node->left = node->right = nullptr;
    return node;
}

int height(AVLNode* node) {
    return node ? node->height : 0;
}

int balance_factor(AVLNode* node) {
    return node ? height(node->left) - height(node->right) : 0;
}
```



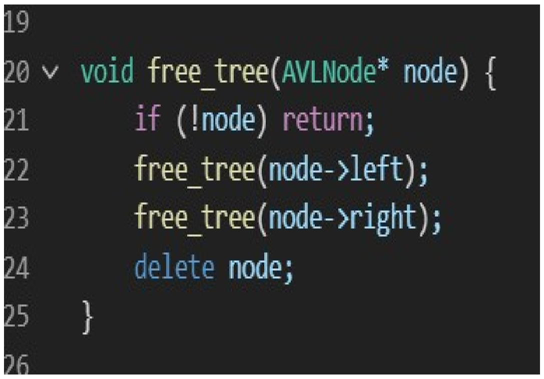
```cpp
void update_height(AVLNode* node) {
    node->height = 1 + std::max(height(node->left), height(node->right));
}

AVLNode* rotate_right(AVLNode* y) {
    AVLNode* x = y->left;
    AVLNode* T2 = x->right;
    x->right = y;
    y->left = T2;
    update_height(y);
    update_height(x);
    return x;
}

AVLNode* rotate_left(AVLNode* x) {
    AVLNode* y = x->right;
    AVLNode* T2 = y->left;
    y->left = x;
    x->right = T2;
    update_height(x);
    update_height(y);
    return y;
}

AVLNode* rebalance(AVLNode* node) {
    update_height(node);
    int bf = balance_factor(node);
    if (bf > 1) {
        if (balance_factor(node->left) < 0)
            node->left = rotate_left(node->left);
        return rotate_right(node);
    }
```



```cpp
AVLNode* insert(AVLNode* node, int key) {
    if (!node) return create_node(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; // Duplicate keys not allowed
    return rebalance(node);
}

AVLNode* find_min(AVLNode* node) {
    return node->left ? find_min(node->left) : node;
}

AVLNode* remove_min(AVLNode* node) {
    if (!node->left)
        return node->right;
    node->left = remove_min(node->left);
    return rebalance(node);
}
```



```cpp
AVLNode* erase(AVLNode* node, int key) {
    if (!node) return nullptr;
    if (key < node->key)
        node->left = erase(node->left, key);
    else if (key > node->key)
        node->right = erase(node->right, key);
    else {
        AVLNode* l = node->left;
        AVLNode* r = node->right;
        delete node;
        if (!r) return l;
        AVLNode* min = find_min(r);
        min->right = remove_min(r);
        min->left = l;
        return rebalance(min);
    }
    return rebalance(node);
}

void inorder(AVLNode* node, const std::function<void(int)>& visit) {
    if (!node) return;
    inorder(node->left, visit);
    visit(node->key);
    inorder(node->right, visit);
}
```



```cpp
void free_tree(AVLNode* node) {
    if (!node) return;
    free_tree(node->left);
    free_tree(node->right);
    delete node;
}
```