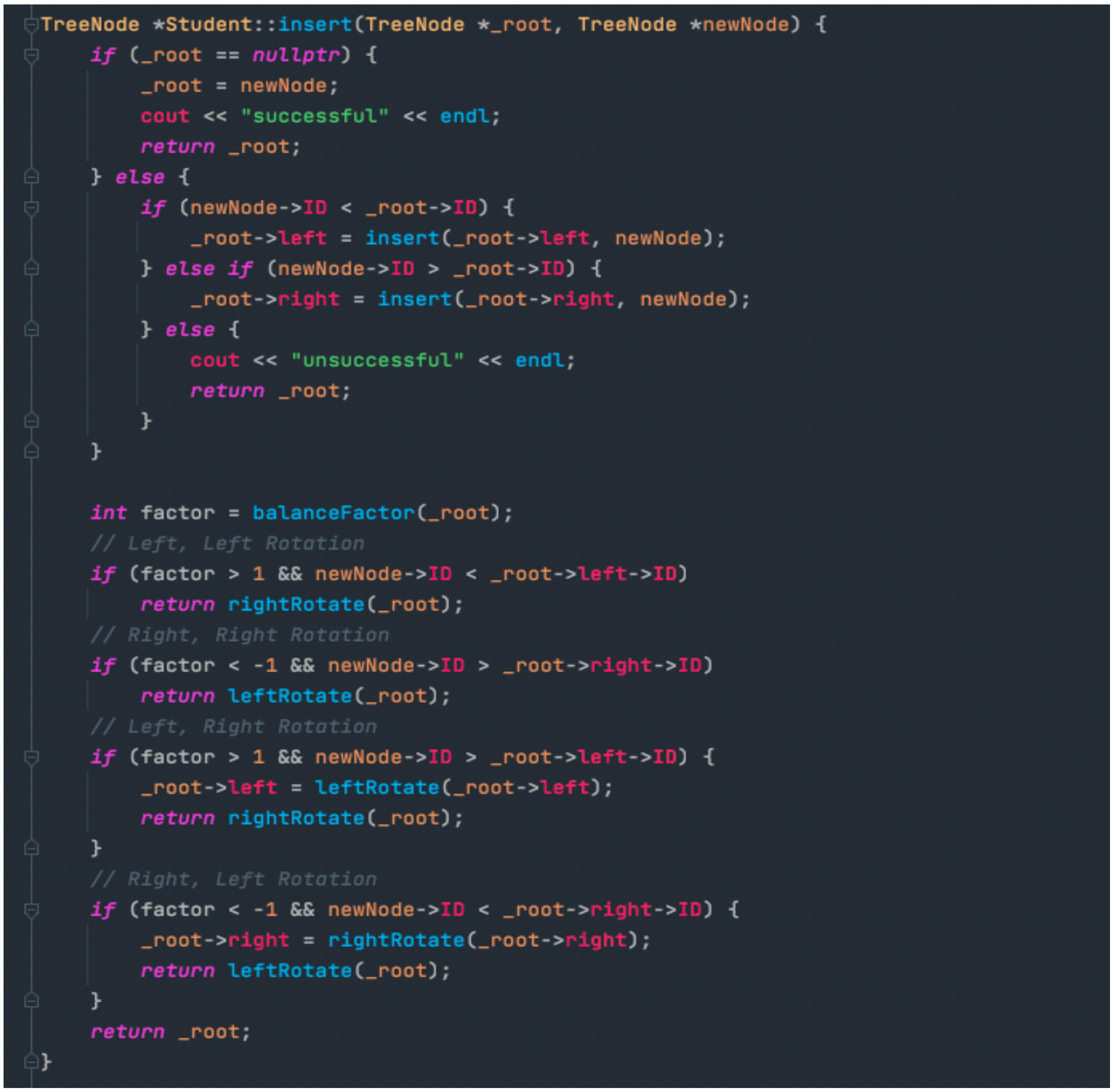


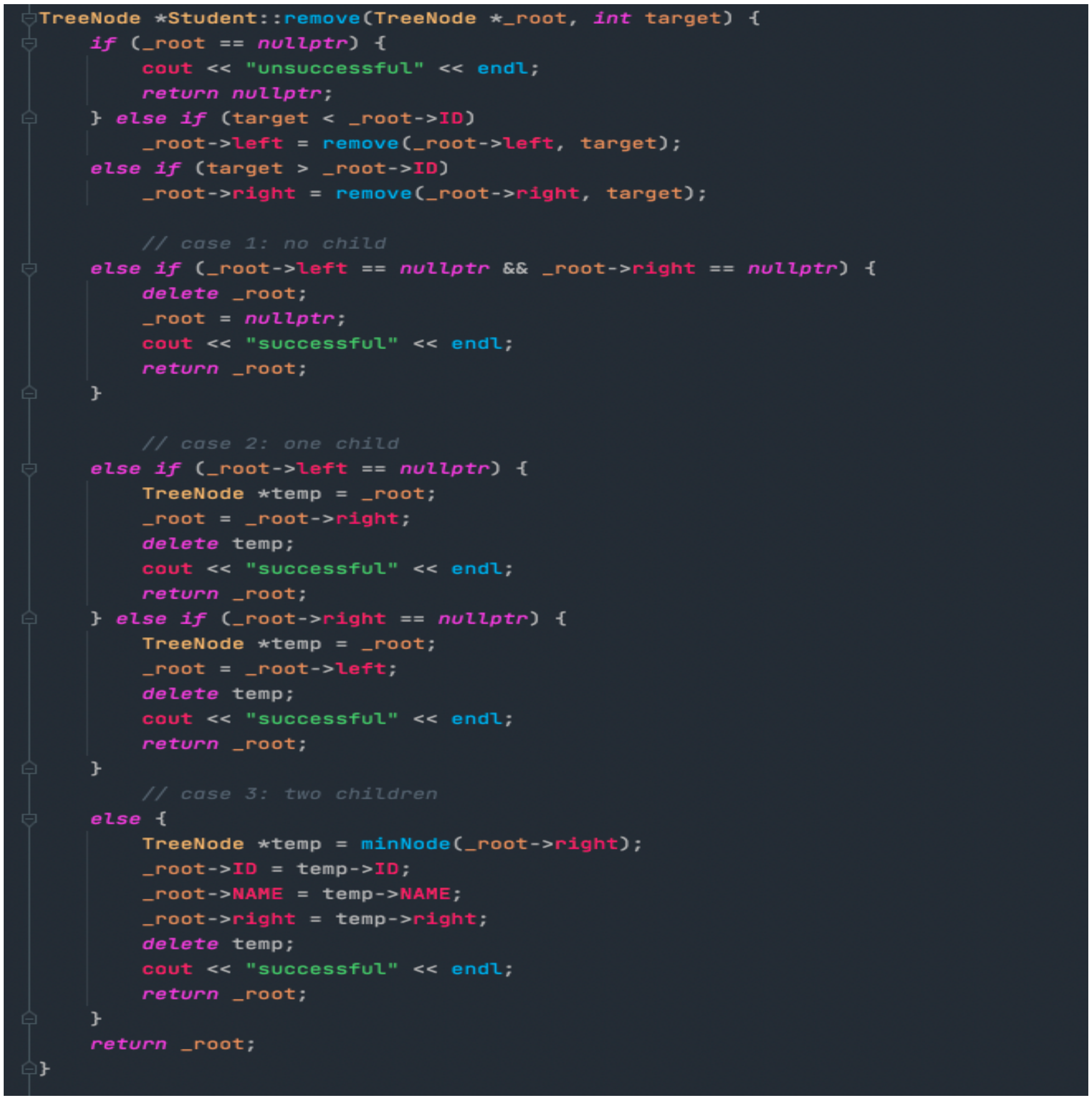
```
TreeNode *Student::insert(TreeNode *_root, TreeNode *newNode) {
    if (_root == nullptr) {
        _root = newNode;
        cout << "successful" << endl;
        return _root;
    } else {
        if (newNode->ID < _root->ID) {
            _root->left = insert(_root->left, newNode);
        } else if (newNode->ID > _root->ID) {
            _root->right = insert(_root->right, newNode);
        } else {
            cout << "unsuccessful" << endl;
            return _root;
        }
    }

    int factor = balanceFactor(_root);
    // Left, Left Rotation
    if (factor > 1 && newNode->ID < _root->left->ID)
        return rightRotate(_root);
    // Right, Right Rotation
    if (factor < -1 && newNode->ID > _root->right->ID)
        return leftRotate(_root);
    // Left, Right Rotation
    if (factor > 1 && newNode->ID > _root->left->ID) {
        _root->left = leftRotate(_root->left);
        return rightRotate(_root);
    }
    // Right, Left Rotation
    if (factor < -1 && newNode->ID < _root->right->ID) {
        _root->right = rightRotate(_root->right);
        return leftRotate(_root);
    }
    return _root;
}
```

It takes log n time to insert a node in a AVL tree

T(n) = log n

where n is the number of nodes in a tree as the recursive calls touch every node

Best Case: O(log n)
Worst Case: O(log n)
Average Case: O(log n)

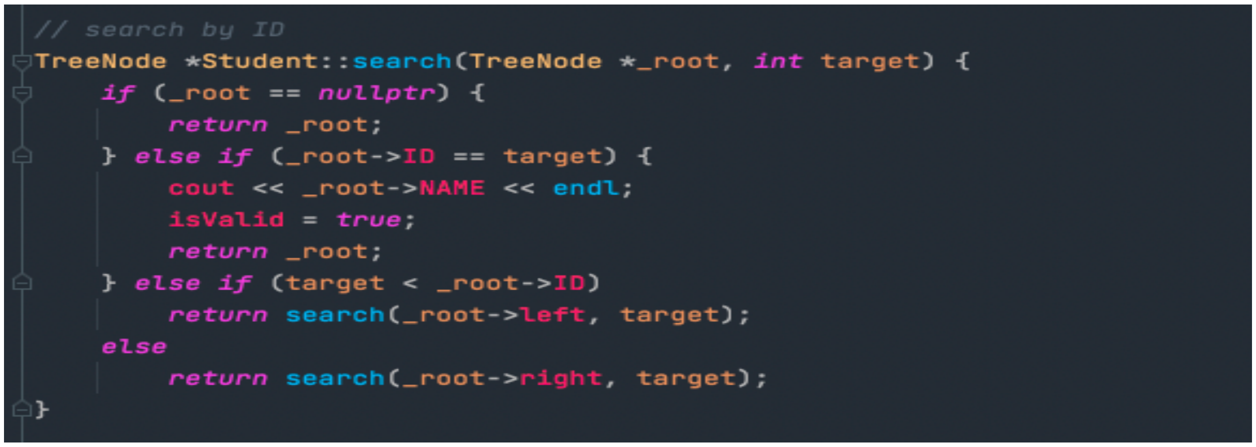

```
TreeNode *Student::remove(TreeNode *_root, int target) {
    if (_root == nullptr) {
        cout << "unsuccessful" << endl;
        return nullptr;
    } else if (target < _root->ID)
        _root->left = remove(_root->left, target);
    else if (target > _root->ID)
        _root->right = remove(_root->right, target);

        // case 1: no child
    else if (_root->left == nullptr && _root->right == nullptr) {
        delete _root;
        _root = nullptr;
        cout << "successful" << endl;
        return _root;
    }

        // case 2: one child
    else if (_root->left == nullptr) {
        TreeNode *temp = _root;
        _root = _root->right;
        delete temp;
        cout << "successful" << endl;
        return _root;
    } else if (_root->right == nullptr) {
        TreeNode *temp = _root;
        _root = _root->left;
        delete temp;
        cout << "successful" << endl;
        return _root;
    }
        // case 3: two children
    else {
        TreeNode *temp = minNode(_root->right);
        _root->ID = temp->ID;
        _root->NAME = temp->NAME;
        _root->right = temp->right;
        delete temp;
        cout << "successful" << endl;
        return _root;
    }
    return _root;
}
```

It takes log n time to remove a node from an AVL tree

T(n) = log n

where n is the number of nodes in a tree as the recursive calls touch every node

Best Case: O(log n)
Worst Case: O(log n)
Average Case: O(log n)



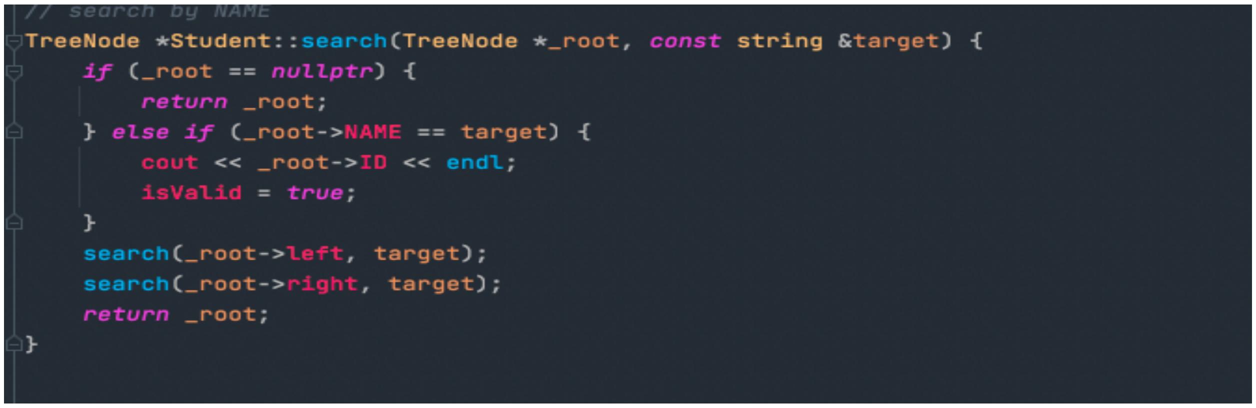
```
// search by ID
TreeNode *Student::search(TreeNode *_root, int target) {
    if (_root == nullptr) {
        return _root;
    } else if (_root->ID == target) {
        cout << _root->NAME << endl;
        isValid = true;
        return _root;
    } else if (target < _root->ID)
        return search(_root->left, target);
    else
        return search(_root->right, target);
}
```

It takes log n time to search a node

T(n) = log n

where n is the number of nodes in a tree as the recursive calls touch every node

Best Case: O(1) (if root node is target then it only will run once)
Worst Case: O(log n)
Average Case: O(log n)



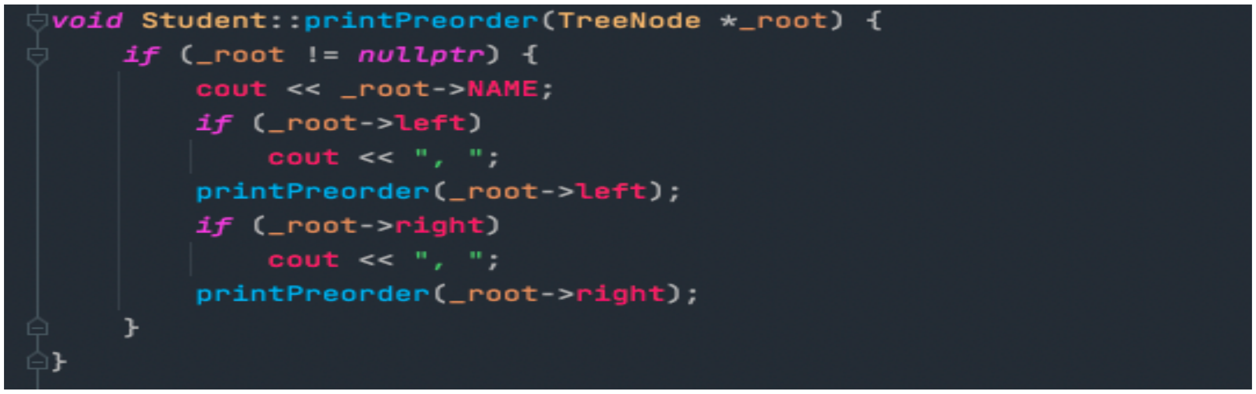
```
// search by NAME
TreeNode *Student::search(TreeNode *_root, const string &target) {
    if (_root == nullptr) {
        return _root;
    } else if (_root->NAME == target) {
        cout << _root->ID << endl;
        isValid = true;
    }
    search(_root->left, target);
    search(_root->right, target);
    return _root;
}
```

It takes n time to search a node because it will traverse all n nodes to check if there is any more same name exits.

T(n) = n

where n is the number of nodes in a tree as the recursive calls touch every node

Best Case: O(n)
Worst Case: O(n)
Average Case: O(n)



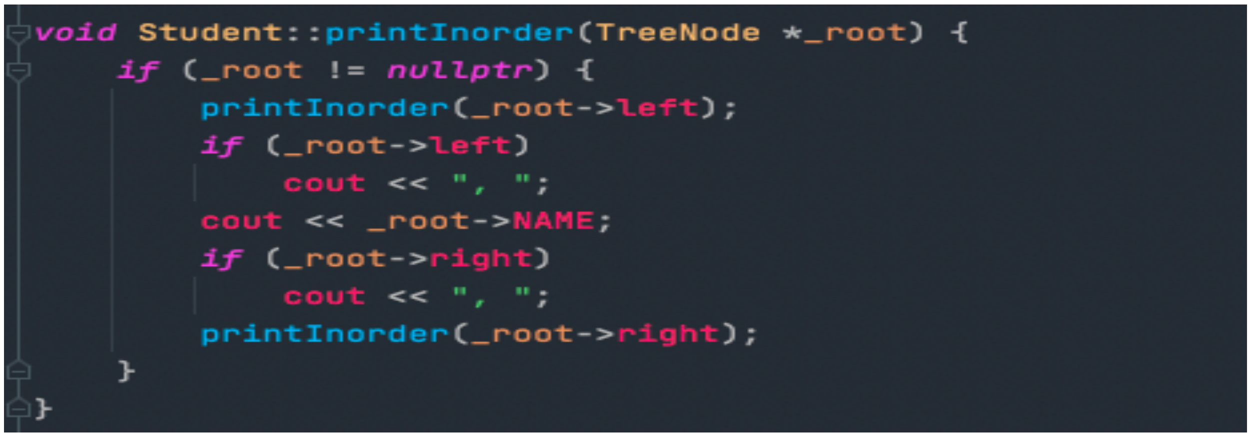
```
void Student::printPreorder(TreeNode *_root) {
    if (_root != nullptr) {
        cout << _root->NAME;
        if (_root->left)
            cout << ", ";
        printPreorder(_root->left);
        if (_root->right)
            cout << ", ";
        printPreorder(_root->right);
    }
}
```

It takes n time to traverse and print all elements in a tree.

T(n) = n

where n is the number of nodes in a tree as the recursive calls touch every node

Best Case: O(n)
Worst Case: O(n)
Average Case: O(n)



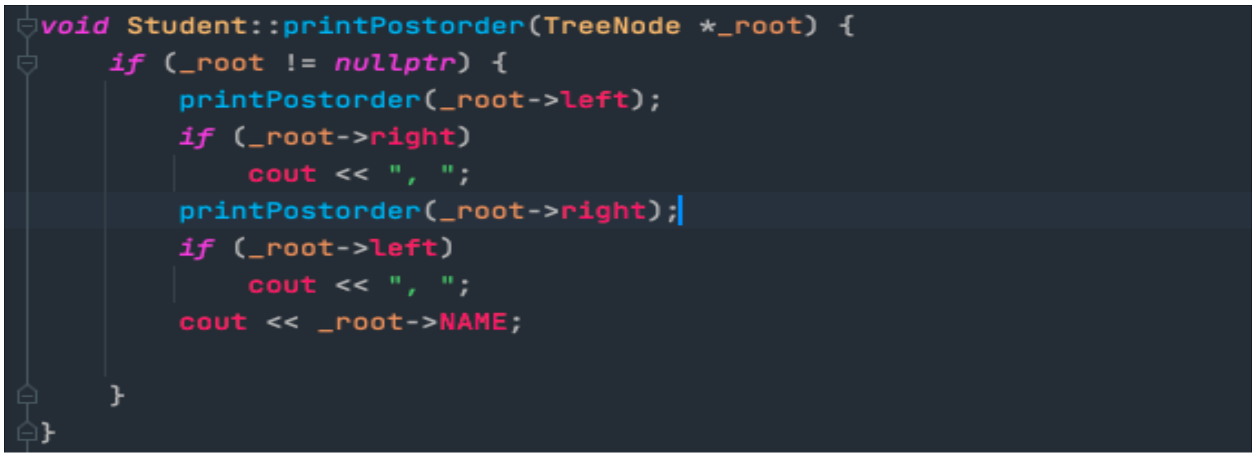
```
void Student::printInorder(TreeNode *_root) {
    if (_root != nullptr) {
        printInorder(_root->left);
        if (_root->left)
            cout << ", ";
        cout << _root->NAME;
        if (_root->right)
            cout << ", ";
        printInorder(_root->right);
    }
}
```

It takes n time to traverse and print all elements in a tree.

T(n) = n

where n is the number of nodes in a tree as the recursive calls touch every node

Best Case: O(n)
Worst Case: O(n)
Average Case: O(n)



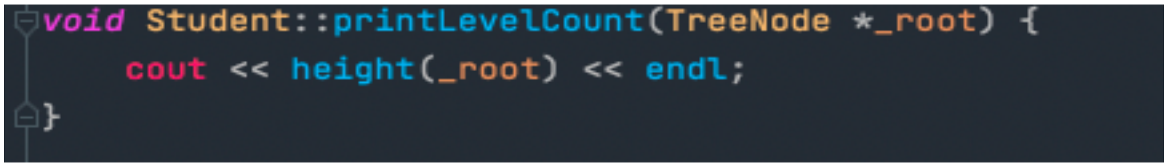
```
void Student::printPostorder(TreeNode *_root) {
    if (_root != nullptr) {
        printPostorder(_root->left);
        if (_root->right)
            cout << ", ";
        printPostorder(_root->right);
        if (_root->left)
            cout << ", ";
        cout << _root->NAME;

    }
}
```

It takes n time to traverse and print all elements in a tree.

T(n) = n

where n is the number of nodes in a tree as the recursive calls touch every node

Best Case: O(n)
Worst Case: O(n)
Average Case: O(n)



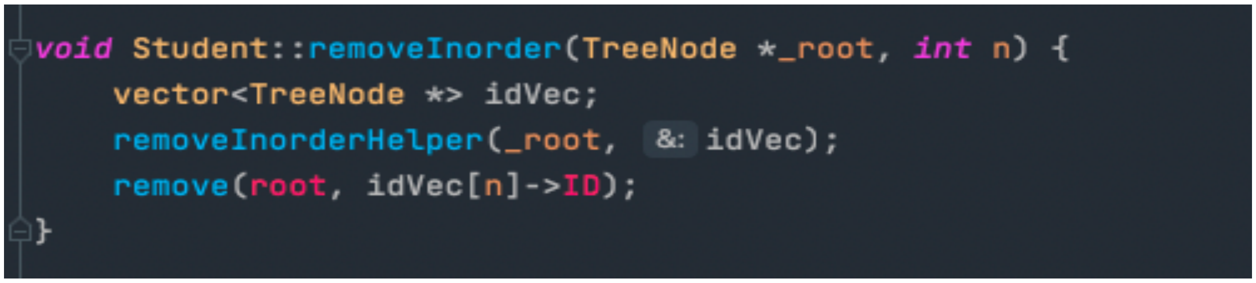
```
void Student::printLevelCount(TreeNode *_root) {
    cout << height(_root) << endl;
}
```

It takes log n time to find a height of an AVL tree

T(n) = log n

where n is the number of nodes in a tree as the recursive calls touch every node

Best Case: O(log n)
Worst Case: O(log n)
Average Case: O(log n)



```
void Student::removeInorder(TreeNode *_root, int n) {
    vector<TreeNode *> idVec;
    removeInorderHelper(_root,  &: idVec);
    remove(root, idVec[n]->ID);
}
```

It takes n time to store all elements in order and takes log n time for deletion.

T(n) = n + log n = n

where n is the number of nodes in a tree as the recursive calls touch every node

Best Case: O(1)
Worst Case: O(1)
Average Case: O(1)

Reflection

From this project, I have learned how BST and AVL tree works. How AVL algorithm makes BST much more efficient. I have learned basic operations of BST such as insertion, deletion, searching, print in order, preorder, and post order. I have learned implementing AVL rotations such as L-L, R-R, L-R, R-L. When a new node gets inserted, then I perform rotation based on balance factor (-1 <= balance factor <= 1) of a node to make BST height balanced tree. At the beginning, understanding and implementing AVL tree was little difficult, but Stepik problems and lectures helped me to understand the concepts. However, If I had to start over to do this same project again, I would implement deletion function as an AVL as well. I would also try to make this project more time and space efficient.