Date: 22/01/2021

Practical 2

2CSDE75 - Advanced Data Structures

Name: Shrey Viradiya

Roll No: 18BCE259

Aim:

Design a balanced binary search tree (AVL) using model 2 (node tree) structure

Code:

Prac2_AVL.cpp

```
// Design a balanced binary search tree (AVL) using
// model 2 (node tree) structure

#include "AVL.h"
#include <iostream>

int main(){
    AVL data("Ajino Motado");

    data.AddData("data.csv", 1);
    data.traverse();
    data.PrettyPrinting();
    std::cout << data.search(1) << std::endl;
    return 0;
}
```

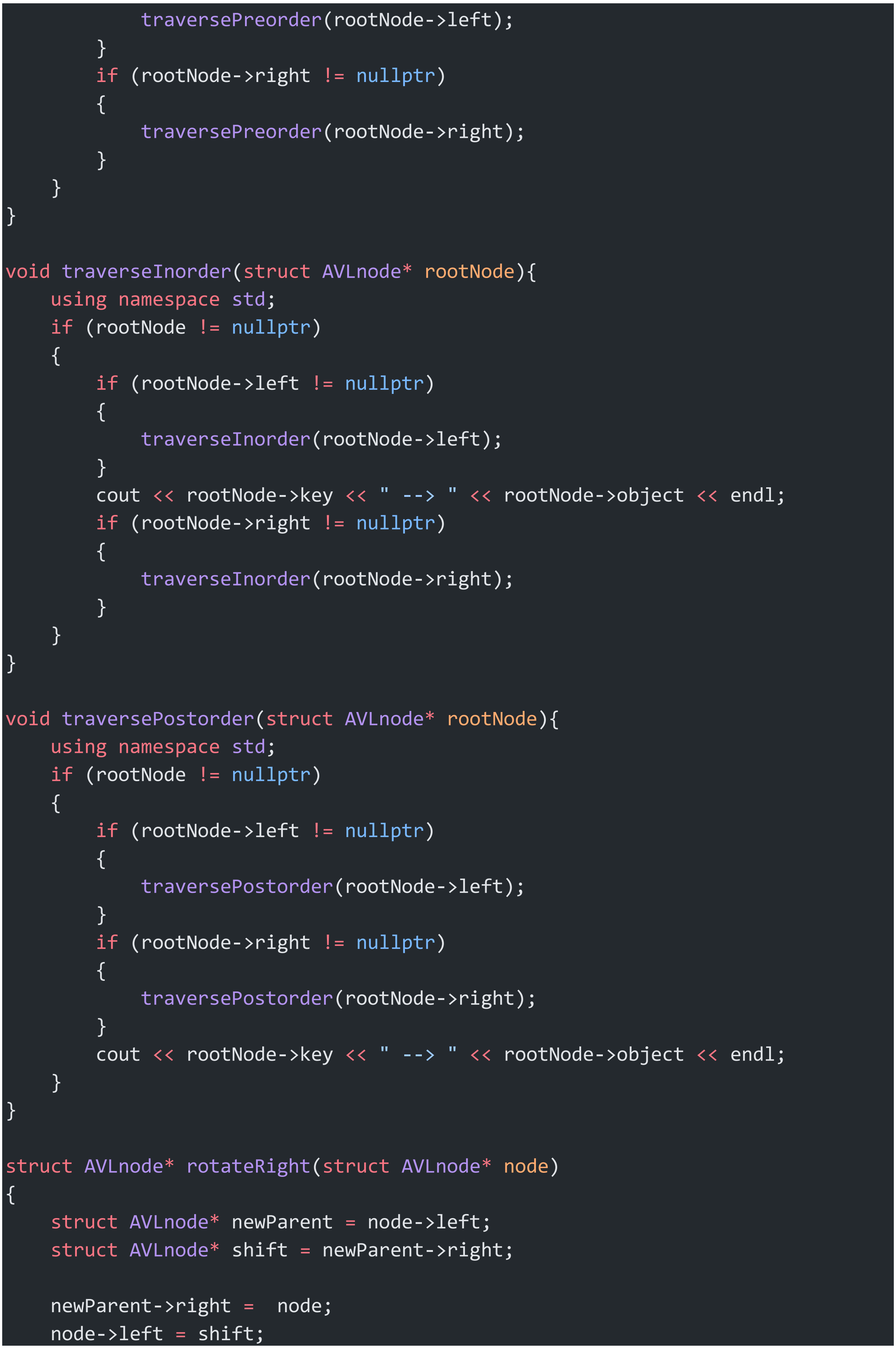
AVL.h

```
#pragma once
#include <iostream>
#include <cstring>
#include <string>
#include <fstream>
#include <vector>
#include <stdexcept> // std::runtime_error
#include <sstream>
#include "AVLutilities.h"
// This header file contains the code for AVL self balancing tree

class AVL
{
    private:
        char name[50];
        struct AVLnode *root;
    public:
        AVL(const char n[50]);
        ~AVL();
        void AddData(std::string filename, int isHeading);
        void insert(int key, int object);
        void traverse(int mode);
        int search(int key);
        void PrettyPrinting();
};
```

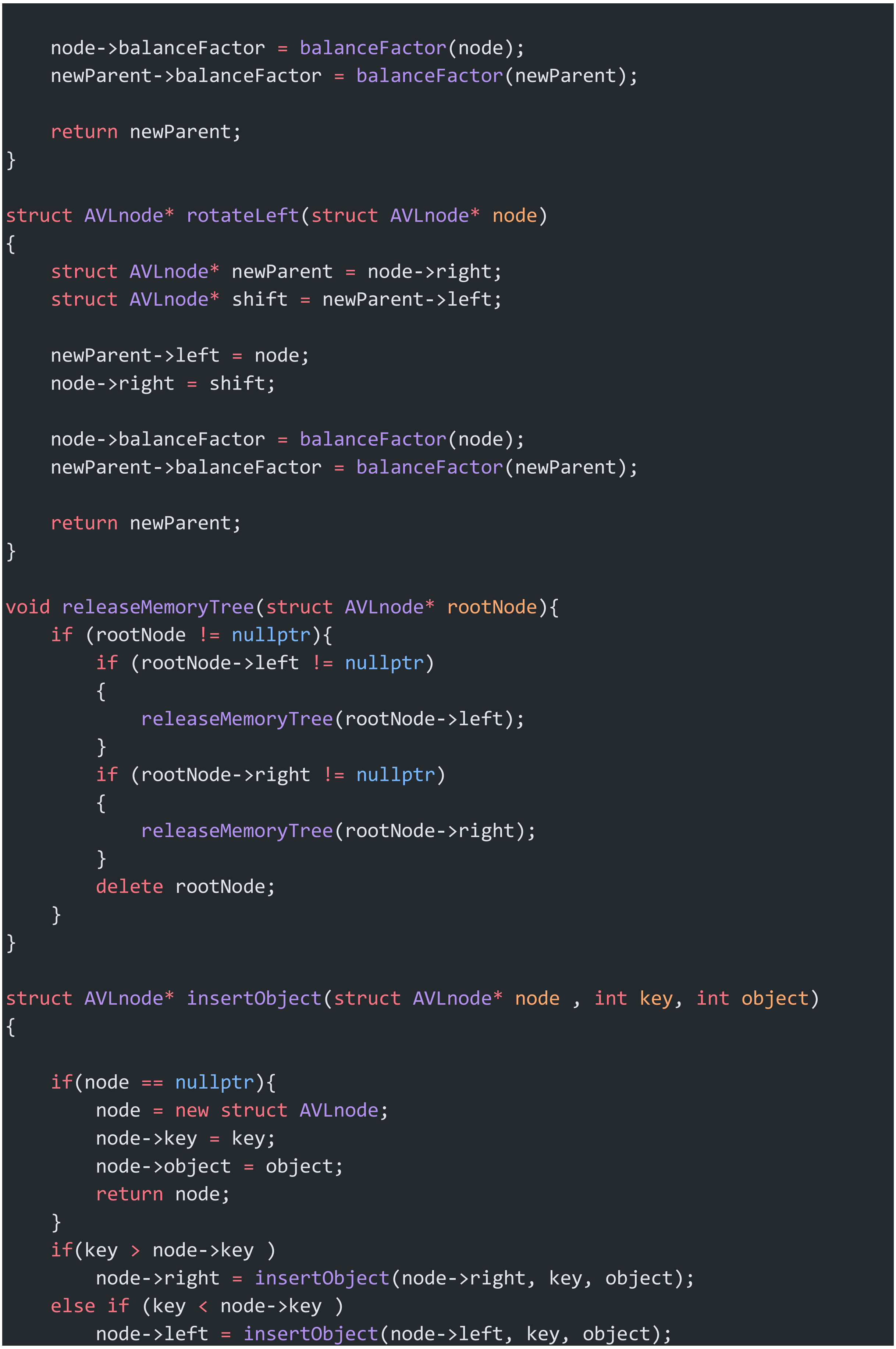
```cpp
AVL::AVL(const char n[50])
{
    strcpy(name, n);
    root = nullptr;
}

AVL::~AVL()
{
    using namespace std;
    releaseMemoryTree(root);
    cout << "Memory Released of " << name << endl;
}

void AVL::insert(int key, int object){
    root = insertObject(root, key, object);
}

int AVL::search(int key){
    struct AVLnode* node = root;

    while (node)
    {
        if (node->key == key)
        {
            return node->object;
        }
        else if(key < node->key){
            node = node->left;
        }
        else{
            node = node->right;
        }
    }
    return 0;
}

void AVL::AddData(std::string filename, int isHeading = 1){
    using namespace std;
    // working with csv in CPP
    // https://www.gormanalysis.com/blog/reading-and-writing-csv-files-with-
cpp/

    ifstream myFile(filename);
    if(!myFile.is_open()) throw runtime_error("Could not open file");

    string line, word;
    int val;
    if (isHeading)  getline(myFile, line);

    // Read data, line by line
    while(getline(myFile, line))
    {
        // Create a stringstream of the current line
        stringstream ss(line);
        pair<int, int> data;

        // add the column data
        // of a row to a pair
        getline(ss, word, ',');
        data.first = stoi(word);

        getline(ss, word, ',');
        data.second = stoi(word);

        insert(data.first, data.second);
    }

    // Close file
    myFile.close();
}

void AVL::traverse(int mode = 1){
    using namespace std;

    cout << "\n\nPrinting The AVL tree: " << name << endl;
    cout << "===========================" << endl;
    cout << "Key --> Value" << endl;
    cout << "===========================" << endl;
    if (mode == 0){
        cout << "Preorder" << endl;
        traversePreorder(root);
    }
    else if (mode == 1){
        cout << "Inorder" << endl;
        traverseInorder(root);
    }
    else if (mode == 2){
        cout << "Postorder" << endl;
        traversePostorder(root);
    }
    else{
        cout << "Invalid Mode" << endl;
        cout << "Inorder" << endl;
        traverseInorder(root);
    }
    cout << "===========================\n" << endl;
}

void AVL::PrettyPrinting(){
    printBT("", root, false);
}
```

AVLutilities.h

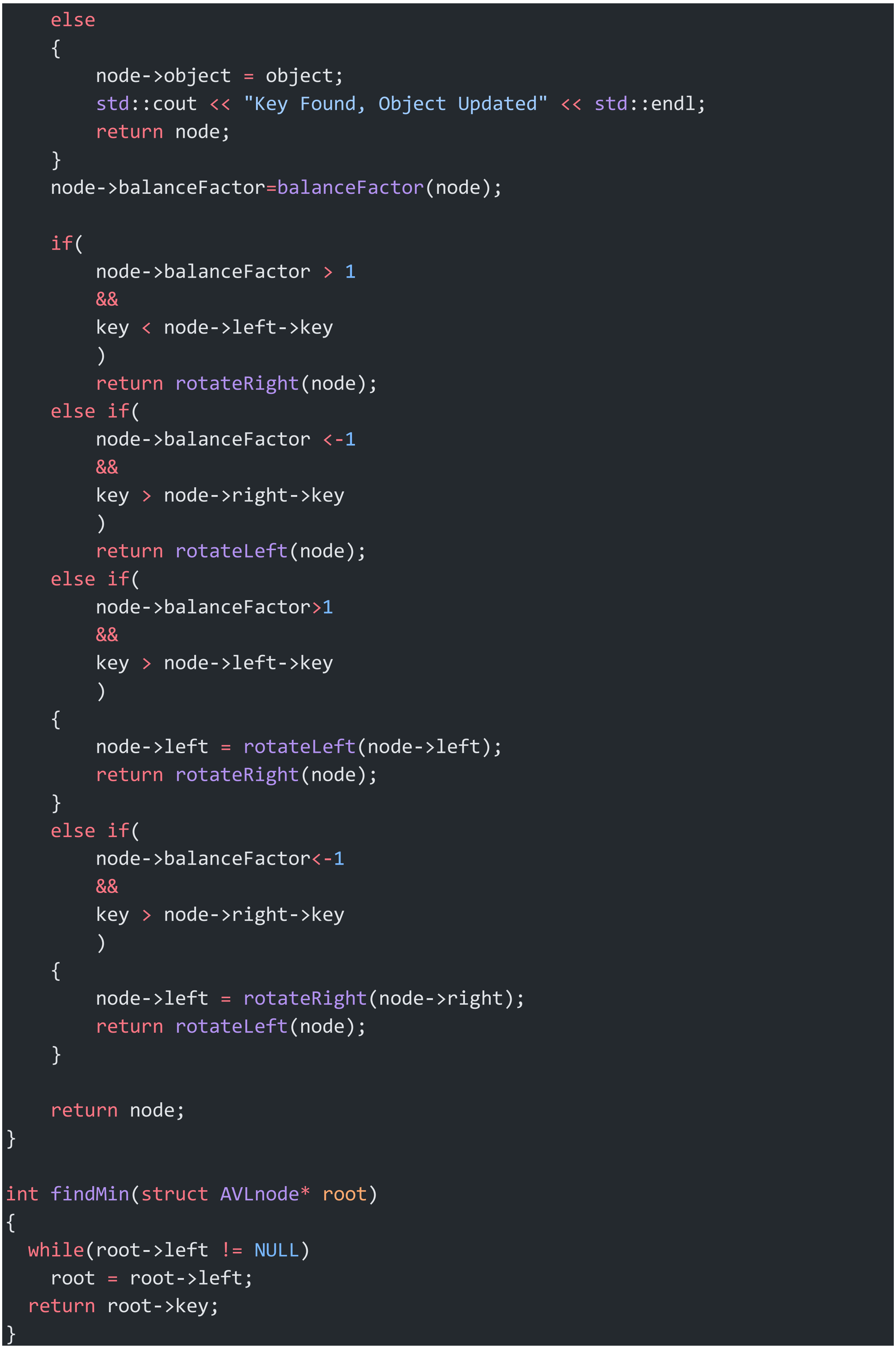
```
#pragma once
#include<iostream>

struct AVLnode
{
    int key;
    int object;
    struct AVLnode *left = nullptr;
    struct AVLnode *right = nullptr;
    int balanceFactor;
};

int height(struct AVLnode* node){
    if(node==nullptr)
        return 0;
    else
    {
        int lh = height(node->left);
        int rh = height(node->right);
        if (lh>rh)
            return lh+1;
        else
            return rh+1;
    }
}

int balanceFactor(struct AVLnode* node)
{
    return (height(node->left)-height(node->right));
}

void traversePreorder(struct AVLnode* rootNode){
    using namespace std;
    if (rootNode != nullptr)
    {
        cout << rootNode->key << " --> " << rootNode->object << endl;
        if (rootNode->left != nullptr)
        {
```



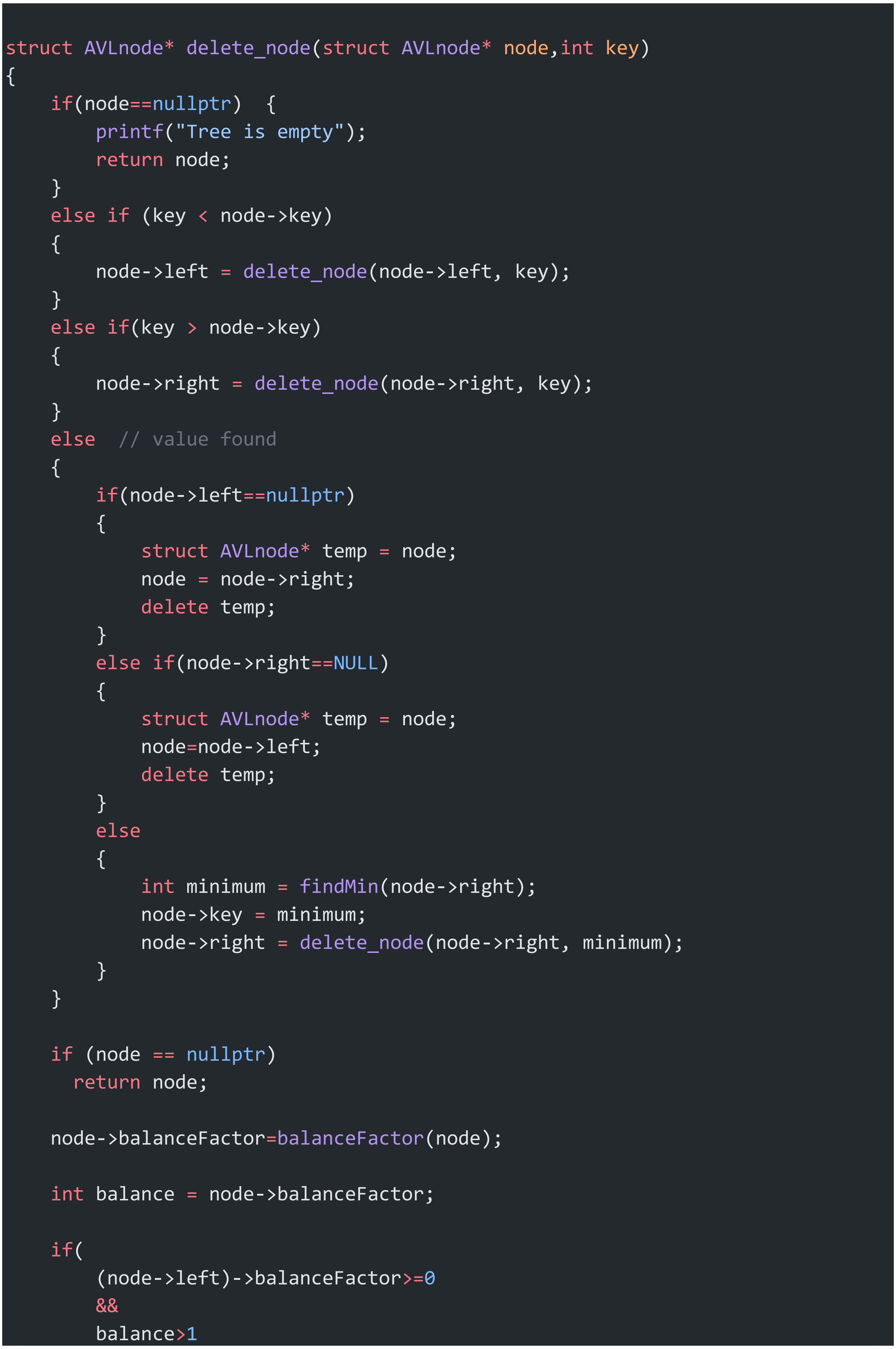
```
            traversePreorder(rootNode->left);
        }
        if (rootNode->right != nullptr)
        {
            traversePreorder(rootNode->right);
        }
    }
}

void traverseInorder(struct AVLnode* rootNode){
    using namespace std;
    if (rootNode != nullptr)
    {
        if (rootNode->left != nullptr)
        {
            traverseInorder(rootNode->left);
        }
        cout << rootNode->key << " --> " << rootNode->object << endl;
        if (rootNode->right != nullptr)
        {
            traverseInorder(rootNode->right);
        }
    }
}

void traversePostorder(struct AVLnode* rootNode){
    using namespace std;
    if (rootNode != nullptr)
    {
        if (rootNode->left != nullptr)
        {
            traversePostorder(rootNode->left);
        }
        if (rootNode->right != nullptr)
        {
            traversePostorder(rootNode->right);
        }
        cout << rootNode->key << " --> " << rootNode->object << endl;
    }
}

struct AVLnode* rotateRight(struct AVLnode* node)
{
    struct AVLnode* newParent = node->left;
    struct AVLnode* shift = newParent->right;

    newParent->right =  node;
    node->left = shift;
```



```
    node->balanceFactor = balanceFactor(node);
    newParent->balanceFactor = balanceFactor(newParent);

    return newParent;
}

struct AVLnode* rotateLeft(struct AVLnode* node)
{
    struct AVLnode* newParent = node->right;
    struct AVLnode* shift = newParent->left;

    newParent->left = node;
    node->right = shift;

    node->balanceFactor = balanceFactor(node);
    newParent->balanceFactor = balanceFactor(newParent);

    return newParent;
}

void releaseMemoryTree(struct AVLnode* rootNode){
    if (rootNode != nullptr){
        if (rootNode->left != nullptr)
        {
            releaseMemoryTree(rootNode->left);
        }
        if (rootNode->right != nullptr)
        {
            releaseMemoryTree(rootNode->right);
        }
        delete rootNode;
    }
}

struct AVLnode* insertObject(struct AVLnode* node , int key, int object)
{

    if(node == nullptr){
        node = new struct AVLnode;
        node->key = key;
        node->object = object;
        return node;
    }
    if(key > node->key )
        node->right = insertObject(node->right, key, object);
    else if (key < node->key )
        node->left = insertObject(node->left, key, object);
```

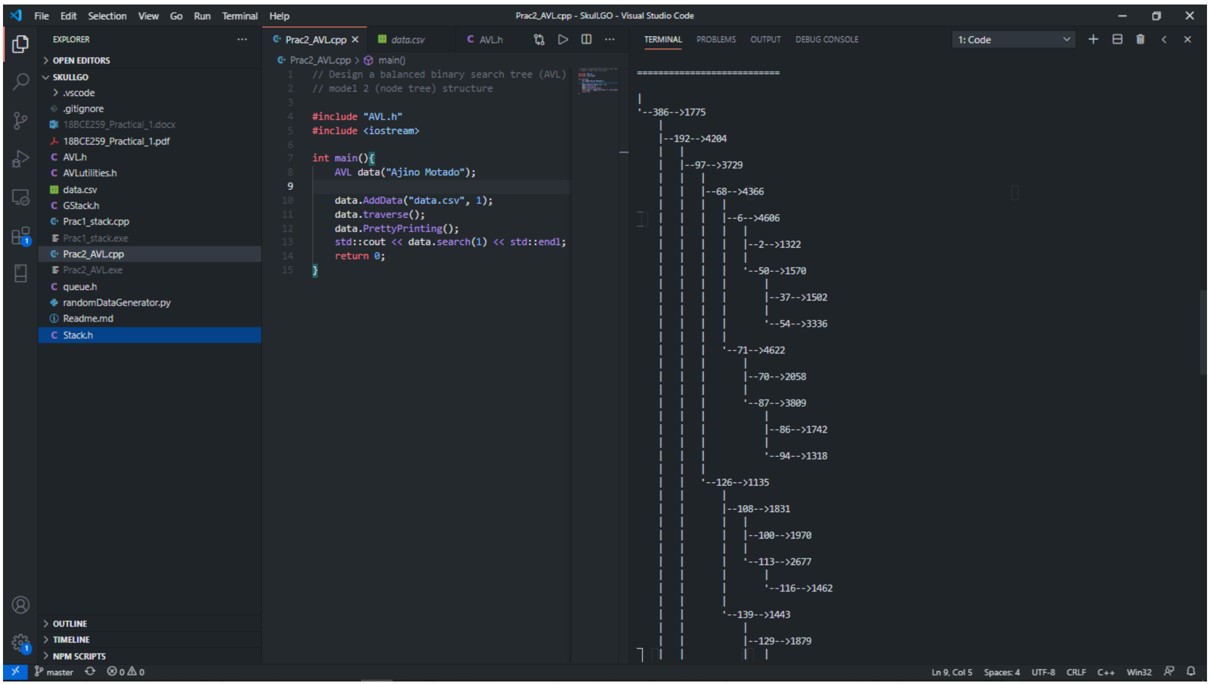


```
    else
    {
        node->object = object;
        std::cout << "Key Found, Object Updated" << std::endl;
        return node;
    }
    node->balanceFactor=balanceFactor(node);

    if(
        node->balanceFactor > 1
        &&
        key < node->left->key
        )
        return rotateRight(node);
    else if(
        node->balanceFactor <-1
        &&
        key > node->right->key
        )
        return rotateLeft(node);
    else if(
        node->balanceFactor>1
        &&
        key > node->left->key
        )
    {
        node->left = rotateLeft(node->left);
        return rotateRight(node);
    }
    else if(
        node->balanceFactor<-1
        &&
        key > node->right->key
        )
    {
        node->left = rotateRight(node->right);
        return rotateLeft(node);
    }

    return node;
}

int findMin(struct AVLnode* root)
{
  while(root->left != NULL)
    root = root->left;
  return root->key;
}
```



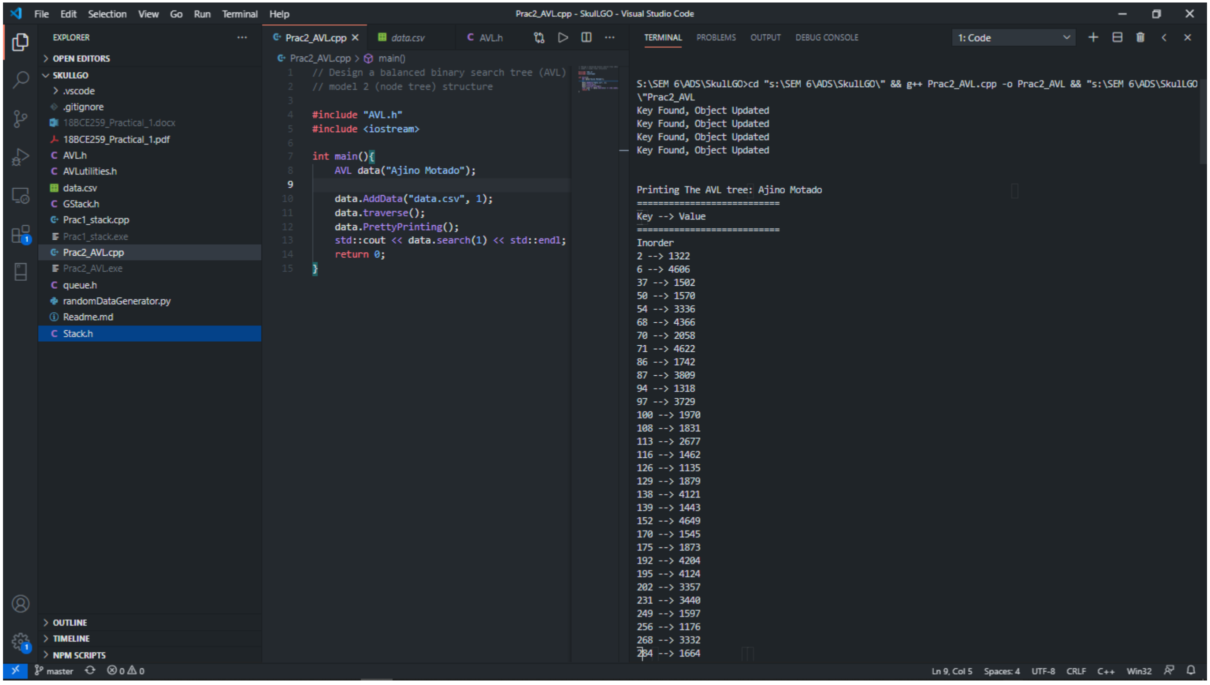
```
struct AVLnode* delete_node(struct AVLnode* node,int key)
{
    if(node==nullptr)  {
        printf("Tree is empty");
        return node;
    }
    else if (key < node->key)
    {
        node->left = delete_node(node->left, key);
    }
    else if(key > node->key)
    {
        node->right = delete_node(node->right, key);
    }
    else  // value found
    {
        if(node->left==nullptr)
        {
            struct AVLnode* temp = node;
            node = node->right;
            delete temp;
        }
        else if(node->right==NULL)
        {
            struct AVLnode* temp = node;
            node=node->left;
            delete temp;
        }
        else
        {
            int minimum = findMin(node->right);
            node->key = minimum;
            node->right = delete_node(node->right, minimum);
        }
    }

    if (node == nullptr)
      return node;

    node->balanceFactor=balanceFactor(node);

    int balance = node->balanceFactor;

    if(
        (node->left)->balanceFactor>=0
        &&
        balance>1
    )
    {
        return rotateRight(node);
    }

    else if(
            (node->right)->balanceFactor<=0
            &&
            balance<-1
        ){
            return rotateLeft(node);
        }

    else if(
        (node->left)->balanceFactor<0
        &&
        balance > 1
        )
        {
            node->left = rotateLeft(node->left);
            return rotateRight(node);
        }
    else if(
        (node->right)->balanceFactor>0
        &&
        balance <-1
        )
        {
            node->left = rotateRight(node->right);
            return rotateLeft(node);
        }
        return node;
}

void printBT(const std::string& prefix, const AVLnode* node, bool isLeft)
{
    if( node != nullptr )
    {
        std::cout << prefix;
        std::cout << "|" << std::endl;
        std::cout << prefix;
        std::cout << (isLeft ? "|--" : "'--" );

        // print the value of the node
        std::cout << node->key << "-->" << node->object << std::endl;

        // enter the next tree level - left and right branch
        printBT( prefix + (isLeft ? "|   " : "    ") , node->left, true);
        printBT( prefix + (isLeft ? "|   " : "    ") , node->right, false);
    }
}
```

Snapshot of the output:



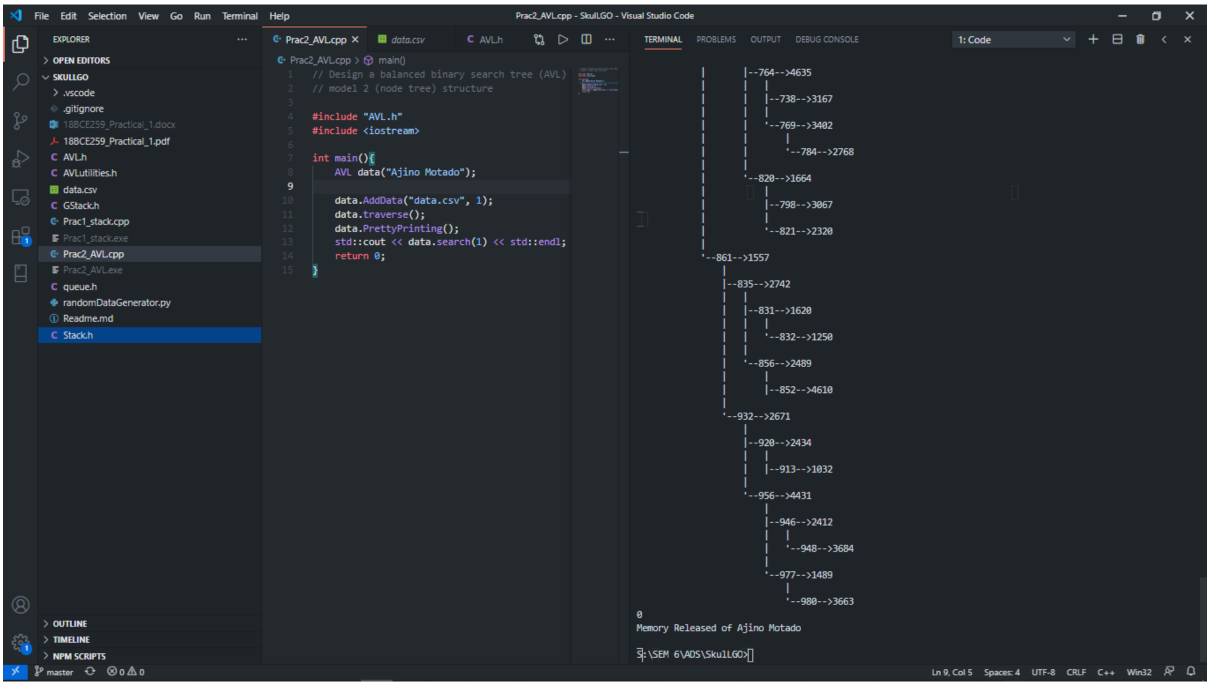
```cpp
// Design a balanced binary search tree (AVL)
// model 2 (node tree) structure

#include "AVL.h"
#include <iostream>

int main(){
    AVL data("Ajino Motado");

    data.AddData("data.csv", 1);
    data.traverse();
    data.PrettyPrinting();
    std::cout << data.search(1) << std::endl;
    return 0;
}
```

```
============================

|
'--386-->1775
    |
    |--192-->4204
    |   |
    |   |--97-->3729
    |   |   |
    |   |   |--68-->4366
    |   |   |   |
    |   |   |   |--6-->4606
    |   |   |   |   |
    |   |   |   |   |--2-->1322
    |   |   |   |   |
    |   |   |   |   '--50-->1570
    |   |   |   |       |
    |   |   |   |       |--37-->1502
    |   |   |   |       |
    |   |   |   |       '--54-->3336
    |   |   |   |
    |   |   |   '--71-->4622
    |   |   |       |
    |   |   |       |--70-->2058
    |   |   |       |
    |   |   |       '--87-->3809
    |   |   |           |
    |   |   |           |--86-->1742
    |   |   |           |
    |   |   |           '--94-->1318
    |   |   |
    |   |   '--126-->1135
    |   |       |
    |   |       |--108-->1831
    |   |       |   |
    |   |       |   |--100-->1970
    |   |       |   |
    |   |       |   '--113-->2677
    |   |       |       |
    |   |       |       '--116-->1462
    |   |       |
    |   |       '--139-->1443
    |   |           |
    |   |           |--129-->1879
    |   |           |   |
```



```cpp
// Design a balanced binary search tree (AVL)
// model 2 (node tree) structure

#include "AVL.h"
#include <iostream>

int main(){
    AVL data("Ajino Motado");

    data.AddData("data.csv", 1);
    data.traverse();
    data.PrettyPrinting();
    std::cout << data.search(1) << std::endl;
    return 0;
}
```

```
S:\SEM 6\ADS\SkulLGO>cd "s:\SEM 6\ADS\SkulLGO\" && g++ Prac2_AVL.cpp -o Prac2_AVL && "s:\SEM 6\ADS\SkulLGO
\"Prac2_AVL
Key Found, Object Updated
Key Found, Object Updated
Key Found, Object Updated
Key Found, Object Updated


Printing The AVL tree: Ajino Motado
============================
Key --> Value
============================
Inorder
2 --> 1322
6 --> 4606
37 --> 1502
50 --> 1570
54 --> 3336
68 --> 4366
70 --> 2058
71 --> 4622
86 --> 1742
87 --> 3809
94 --> 1318
97 --> 3729
100 --> 1970
108 --> 1831
113 --> 2677
116 --> 1462
126 --> 1135
129 --> 1879
138 --> 4121
139 --> 1443
152 --> 4649
170 --> 1545
175 --> 1873
192 --> 4204
195 --> 4124
202 --> 3357
231 --> 3440
249 --> 1597
256 --> 1176
268 --> 3332
284 --> 1664
```



```cpp
// Design a balanced binary search tree (AVL)
// model 2 (node tree) structure

#include "AVL.h"
#include <iostream>

int main(){
    AVL data("Ajino Motado");

    data.AddData("data.csv", 1);
    data.traverse();
    data.PrettyPrinting();
    std::cout << data.search(1) << std::endl;
    return 0;
}
```

```
            |       |--764-->4635
            |       |   |
            |       |   |--738-->3167
            |       |   |
            |       |   '--769-->3402
            |       |       |
            |       |       '--784-->2768
            |       |
            |       '--820-->1664
            |           |
            |           |--798-->3067
            |           |
            |           '--821-->2320
            |
            '--861-->1557
                |
                |--835-->2742
                |   |
                |   |--831-->1620
                |   |   |
                |   |   '--832-->1250
                |   |
                |   '--856-->2489
                |       |
                |       |--852-->4610
                |
                '--932-->2671
                    |
                    |--920-->2434
                    |   |
                    |   |--913-->1032
                    |
                    '--956-->4431
                        |
                        |--946-->2412
                        |   |
                        |   '--948-->3684
                        |
                        '--977-->1489
                            |
                            '--980-->3663
0
Memory Released of Ajino Motado

S:\SEM 6\ADS\SkullGO>
```

Conclusion:

The AVL tree is faster than the BST. When we know that we have to do more searching than insertions, we should without any doubt use AVL trees.