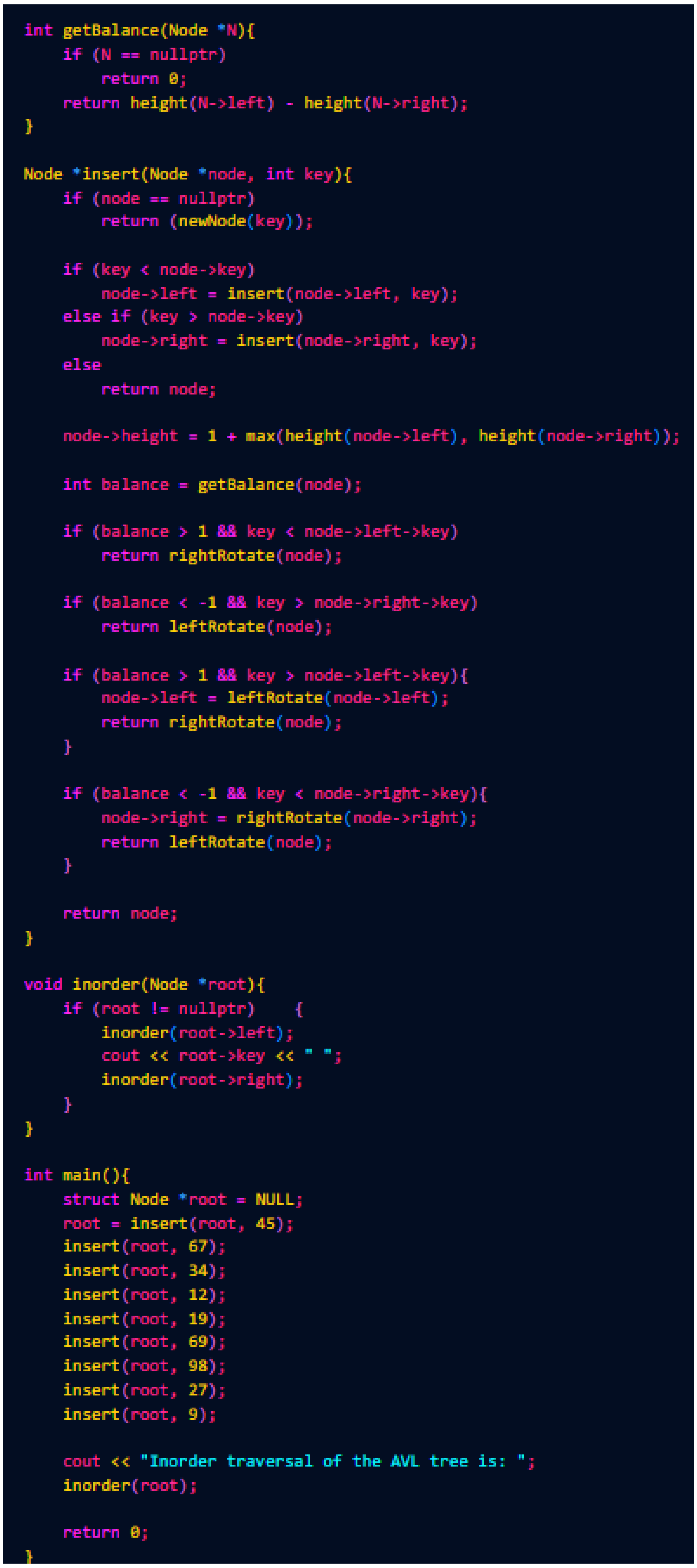
LAB-11-ASSIGNMENT

Name: K V Jaya Harsha

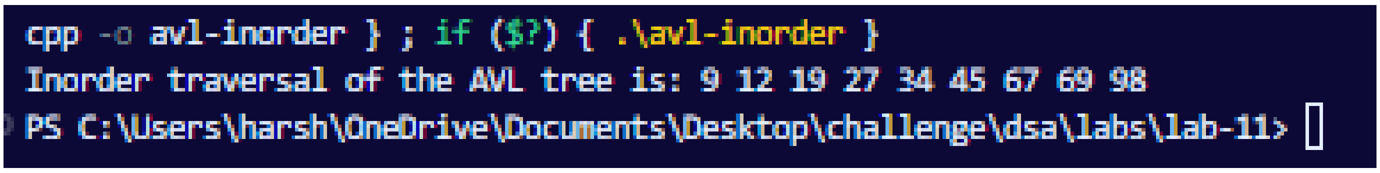
Roll no: CS23B1034 Date: 16-10-2024

Q1. AVL tree (in cpp)

```cpp
// K V Jaya Harsha
// CS23B1034
#include <iostream>
using namespace std;

struct Node{
    int key;
    Node *left;
    Node *right;
    int height;
};

int height(Node *N){
    if (N == nullptr)
        return 0;
    return N->height;
}

int max(int a, int b){
    return (a > b) ? a : b;
}

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

Node *rightRotate(Node *y){
    Node *x = y->left;
    Node *T2 = x->right;

    x->right = y;
    y->left = T2;

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

    return x;
}

Node *leftRotate(Node *x){
    Node *y = x->right;
    Node *T2 = y->left;

    y->left = x;
    x->right = T2;

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

    return y;
}
```

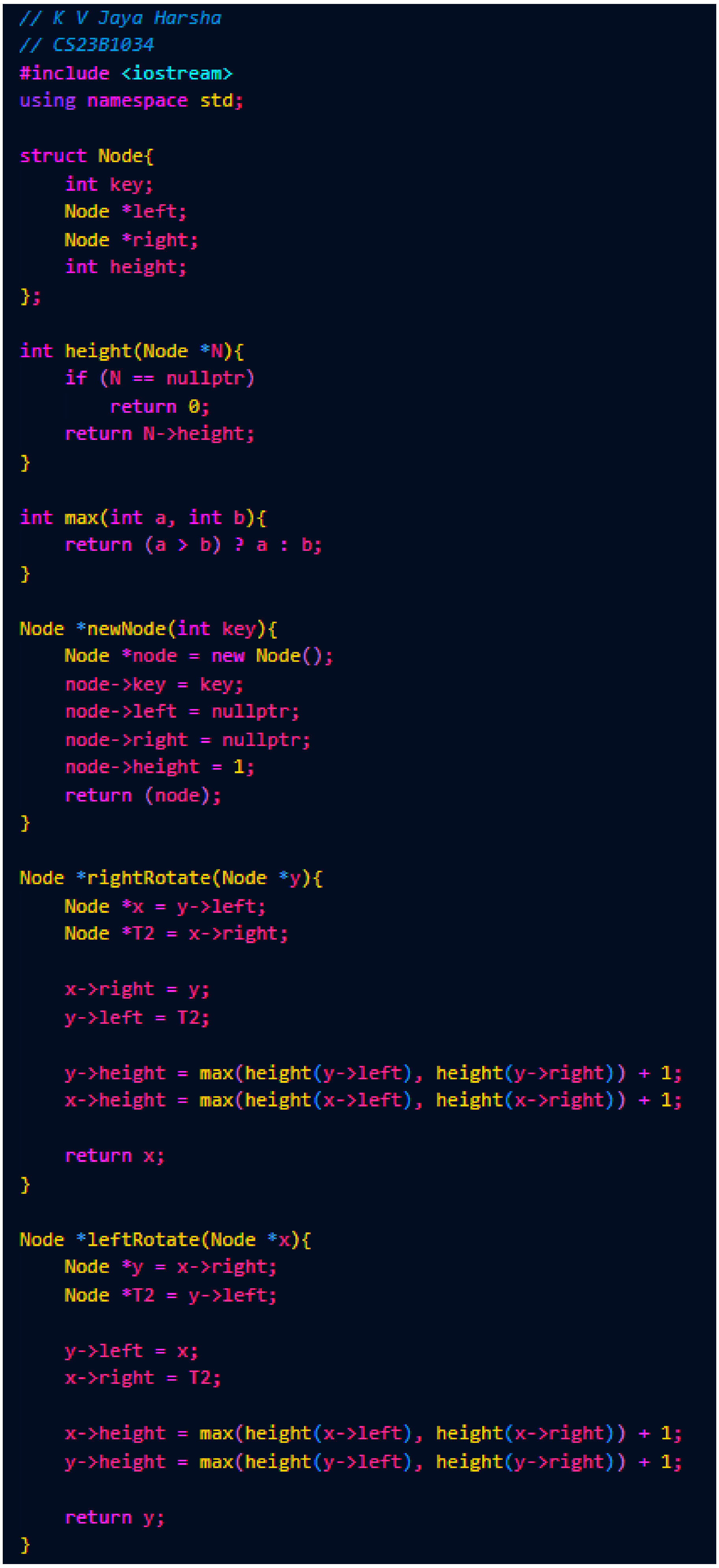


```cpp
int getBalance(Node *N){
    if (N == nullptr)
        return 0;
    return height(N->left) - height(N->right);
}

Node *insert(Node *node, int key){
    if (node == nullptr)
        return (newNode(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;

    node->height = 1 + max(height(node->left), height(node->right));

    int balance = getBalance(node);

    if (balance > 1 && key < node->left->key)
        return rightRotate(node);

    if (balance < -1 && key > node->right->key)
        return leftRotate(node);

    if (balance > 1 && key > node->left->key){
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    if (balance < -1 && key < node->right->key){
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
}

void inorder(Node *root){
    if (root != nullptr)    {
        inorder(root->left);
        cout << root->key << " ";
        inorder(root->right);
    }
}

int main(){
    struct Node *root = NULL;
    root = insert(root, 45);
    insert(root, 67);
    insert(root, 34);
    insert(root, 12);
    insert(root, 19);
    insert(root, 69);
    insert(root, 98);
    insert(root, 27);
    insert(root, 9);

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

    return 0;
}
```

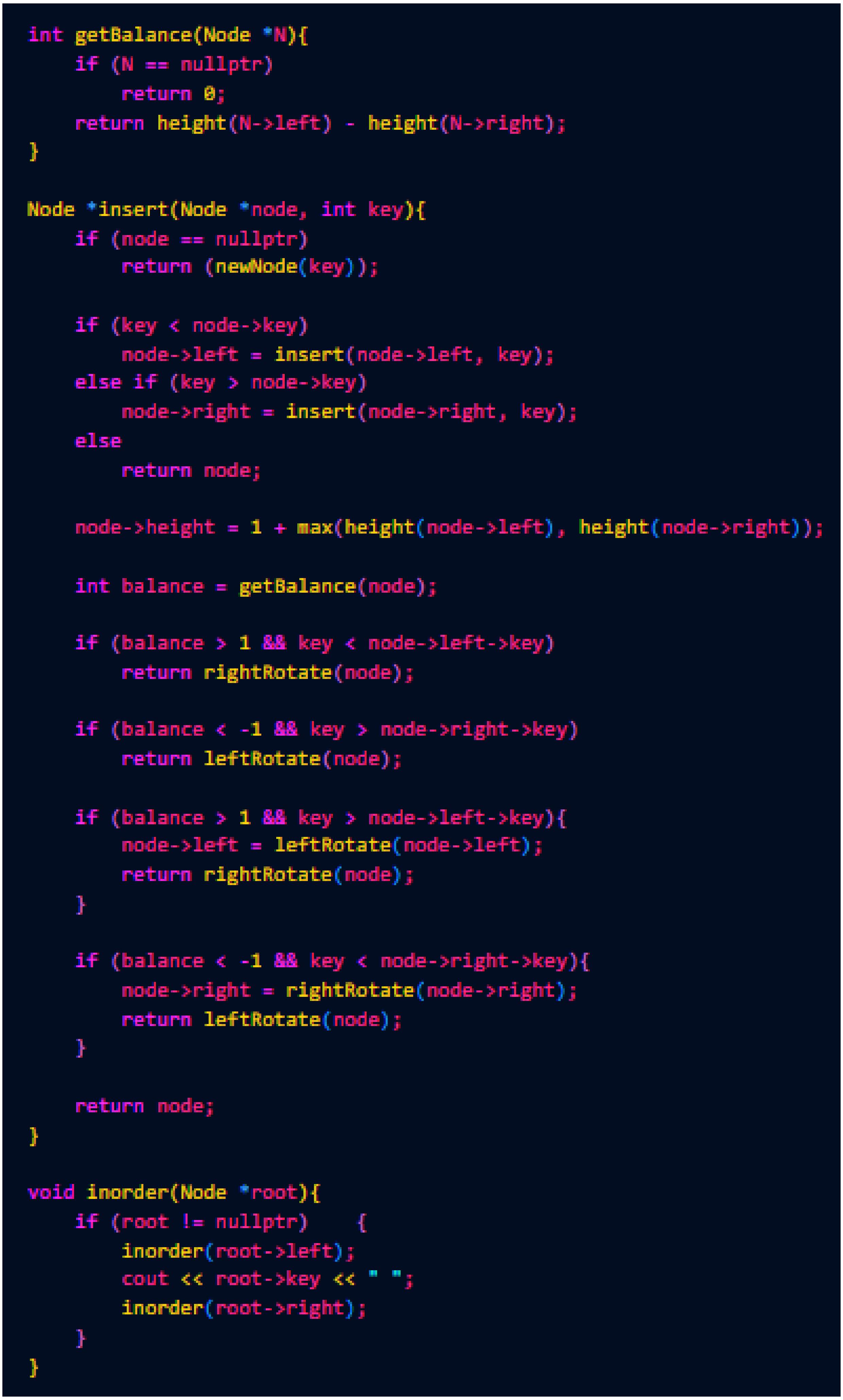


```
cpp -o avl-inorder } ; if ($?) { .\avl-inorder }
Inorder traversal of the AVL tree is: 9 12 19 27 34 45 67 69 98
PS C:\Users\harsh\OneDrive\Documents\Desktop\challenge\dsa\labs\lab-11>
```

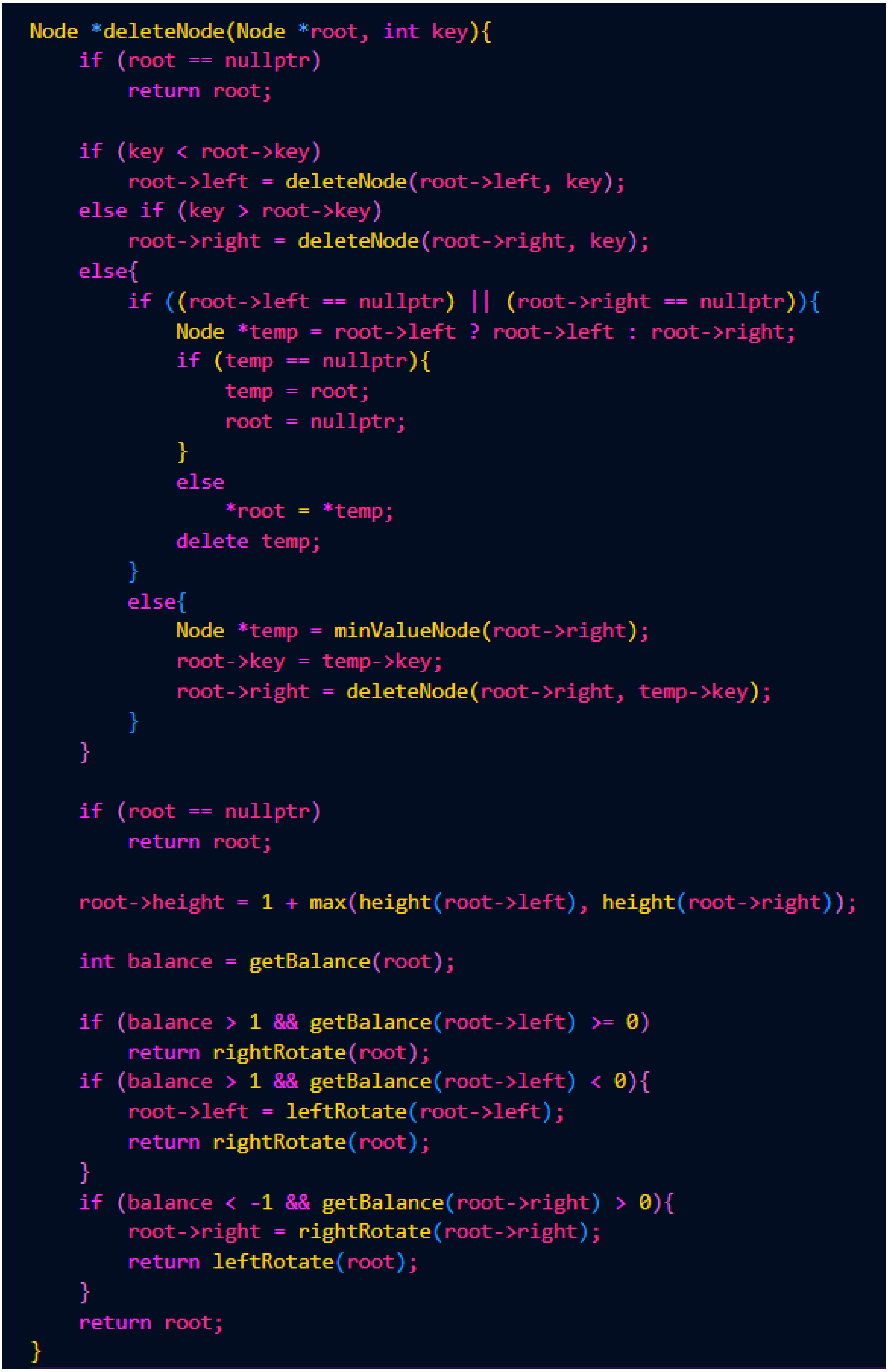
Q2. Delete three nodes one by one from AVL tree (in cpp)



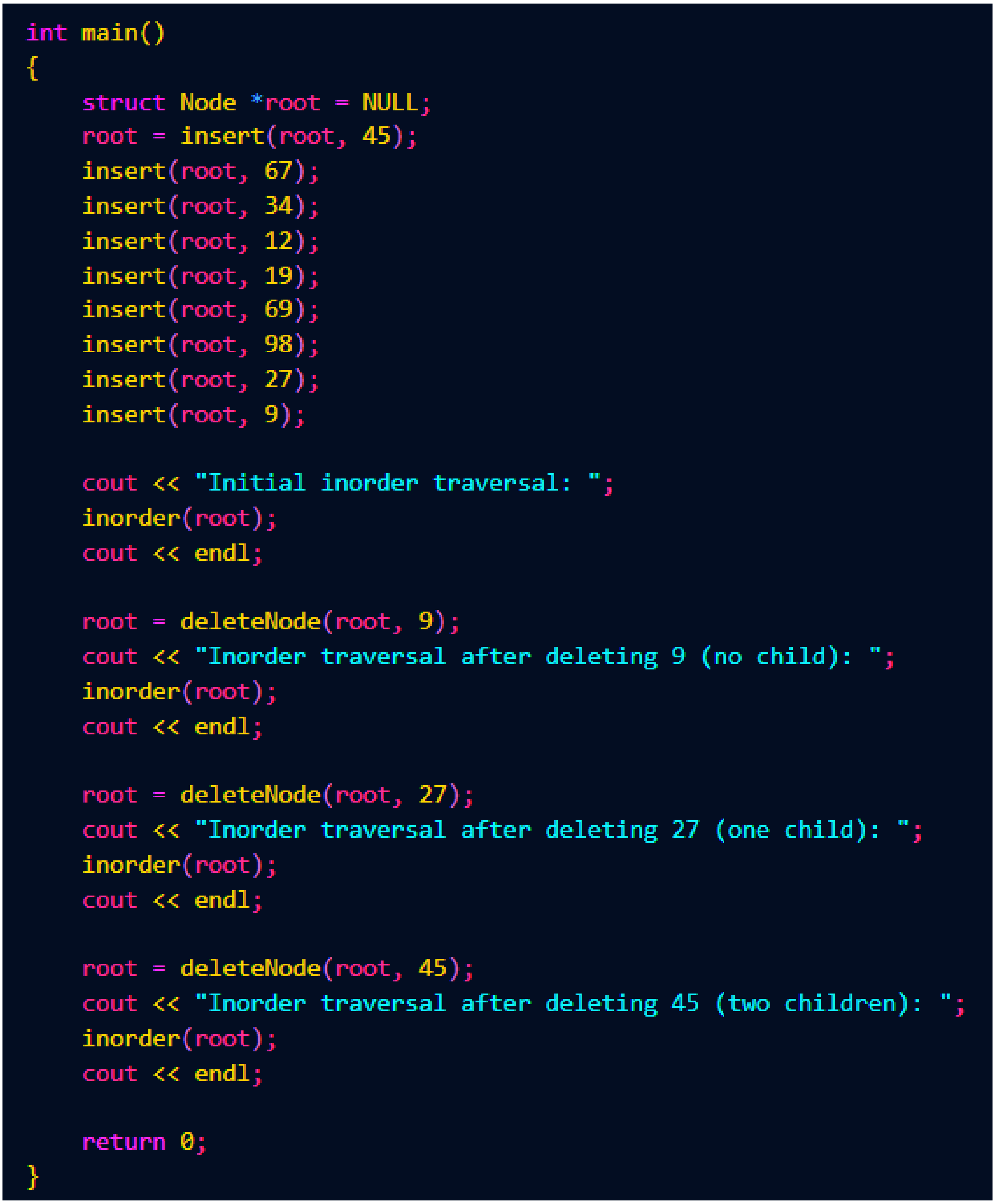
```cpp
// K V Jaya Harsha
// CS23B1034
#include <iostream>
using namespace std;

struct Node{
    int key;
    Node *left;
    Node *right;
    int height;
};

int height(Node *N){
    if (N == nullptr)
        return 0;
    return N->height;
}

int max(int a, int b){
    return (a > b) ? a : b;
}

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

Node *rightRotate(Node *y){
    Node *x = y->left;
    Node *T2 = x->right;

    x->right = y;
    y->left = T2;

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

    return x;
}

Node *leftRotate(Node *x){
    Node *y = x->right;
    Node *T2 = y->left;

    y->left = x;
    x->right = T2;

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

    return y;
}
```



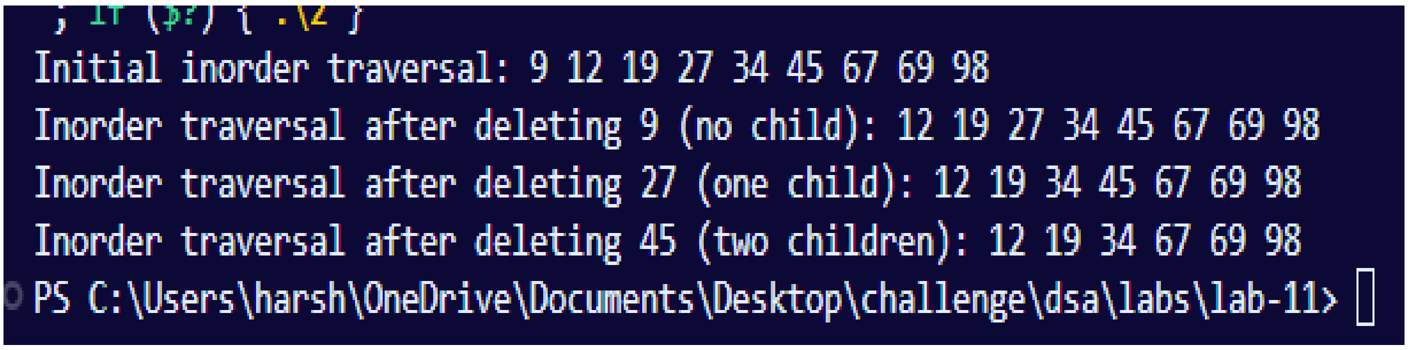
```
int getBalance(Node *N){
    if (N == nullptr)
        return 0;
    return height(N->left) - height(N->right);
}

Node *insert(Node *node, int key){
    if (node == nullptr)
        return (newNode(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;

    node->height = 1 + max(height(node->left), height(node->right));

    int balance = getBalance(node);

    if (balance > 1 && key < node->left->key)
        return rightRotate(node);

    if (balance < -1 && key > node->right->key)
        return leftRotate(node);

    if (balance > 1 && key > node->left->key){
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }

    if (balance < -1 && key < node->right->key){
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }

    return node;
}

void inorder(Node *root){
    if (root != nullptr)    {
        inorder(root->left);
        cout << root->key << " ";
        inorder(root->right);
    }
}
```



```cpp
Node *deleteNode(Node *root, int key){
    if (root == nullptr)
        return root;

    if (key < root->key)
        root->left = deleteNode(root->left, key);
    else if (key > root->key)
        root->right = deleteNode(root->right, key);
    else{
        if ((root->left == nullptr) || (root->right == nullptr)){
            Node *temp = root->left ? root->left : root->right;
            if (temp == nullptr){
                temp = root;
                root = nullptr;
            }
            else
                *root = *temp;
            delete temp;
        }
        else{
            Node *temp = minValueNode(root->right);
            root->key = temp->key;
            root->right = deleteNode(root->right, temp->key);
        }
    }

    if (root == nullptr)
        return root;

    root->height = 1 + max(height(root->left), height(root->right));

    int balance = getBalance(root);

    if (balance > 1 && getBalance(root->left) >= 0)
        return rightRotate(root);
    if (balance > 1 && getBalance(root->left) < 0){
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }
    if (balance < -1 && getBalance(root->right) > 0){
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }
    return root;
}
```



```cpp
int main()
{
    struct Node *root = NULL;
    root = insert(root, 45);
    insert(root, 67);
    insert(root, 34);
    insert(root, 12);
    insert(root, 19);
    insert(root, 69);
    insert(root, 98);
    insert(root, 27);
    insert(root, 9);

    cout << "Initial inorder traversal: ";
    inorder(root);
    cout << endl;

    root = deleteNode(root, 9);
    cout << "Inorder traversal after deleting 9 (no child): ";
    inorder(root);
    cout << endl;

    root = deleteNode(root, 27);
    cout << "Inorder traversal after deleting 27 (one child): ";
    inorder(root);
    cout << endl;

    root = deleteNode(root, 45);
    cout << "Inorder traversal after deleting 45 (two children): ";
    inorder(root);
    cout << endl;

    return 0;
}
```



```
 ; if ($?) { .\2 }
Initial inorder traversal: 9 12 19 27 34 45 67 69 98
Inorder traversal after deleting 9 (no child): 12 19 27 34 45 67 69 98
Inorder traversal after deleting 27 (one child): 12 19 34 45 67 69 98
Inorder traversal after deleting 45 (two children): 12 19 34 67 69 98
PS C:\Users\harsh\OneDrive\Documents\Desktop\challenge\dsa\labs\lab-11>
```