CODE 2:

#include <iostream>

using namespace std;

#include <iostream>

using namespace std;

class node

{

public:

    int data;

    node \*left, \*right;

    node()

    {

        data = 0;

        left = NULL;

        right = NULL;

    }

    node(int data)

    {

        this->data = data;

        left = NULL;

        right = NULL;

    }

};

class BST

{

public:

    node \*root, \*n1;

    node \*parent;

    BST()

    {

        root = NULL;

    }

    node \*rightRotate(node \*y)

    {

        node \*x = y->left;

        node \*T2 = x->right;

        x->right = y;

        y->left = T2;

        return x;

    }

    node \*leftRotate(node \*x)

    {

        node \*y = x->right;

        node \*T2 = y->left;

        y->left = x;

        x->right = T2;

        return y;

    }

    node \*insert(int data)

    {

        n1 = new node(data);

        if (root == NULL)

        {

            root = n1;

            return root;

        }

        else

        {

            node \*current = root;

            parent = NULL;

            while (true)

            {

                if (data > current->data)

                {

                    parent = current;

                    if (current->right == NULL)

                    {

                        current->right = n1;

                        break;

                    }

                    current = current->right;

                }

                else

                {

                    parent = current;

                    if (current->left == NULL)

                    {

                        current->left = n1;

                        break;

                    }

                    current = current->left;

                }

            }

        }

        int getbalance = balancefactor(root);

        if (getbalance > 1 && data < parent->data)

            return rightRotate(parent);

        if (getbalance < -1 && data > parent->data)

            return leftRotate(parent);

        if (getbalance > 1 && data < parent->data)

        {

            parent->left = leftRotate(parent->left);

            return rightRotate(parent);

        }

        if (getbalance < -1 && data > parent->data)

        {

            parent->right = rightRotate(parent->right);

            return leftRotate(parent);

        }

        if (data > parent->data)

            parent->right = n1;

        else

            parent->left = n1;

        return root;

    }

    int height(node \*r)

    {

        if (r == NULL)

        {

            return 0;

        }

        int lheight = height(r->left);

        int rheight = height(r->right);

        if (lheight > rheight)

            return (lheight + 1);

        else

            return (rheight + 1);

    }

    int balancefactor(node \*ro)

    {

        int lh, rh;

        if (ro == NULL)

        {

            return 1;

        }

        lh = height(ro->left);

        rh = height(ro->right);

        int result = lh - rh;

        // cout << "result is " << result<<endl;

        return result;

    }

    void display(node \*n)

    {

        if (n == nullptr)

            return;

        display(n->left);

        cout << n->data << "\t";

        display(n->right);

    }

    ~BST()

    {

        delete n1;

    }

};

int main()

{

    int option;

    BST tree;

    do

    {

        cout << "What do you want to do?\n";

        cout << "1: For inserting value in a BST\n";

        cout << "2: For Ending the program\n";

        cin >> option;

        if (option == 1)

        {

            int val;

            cout << "Enter the value you want to insert\n";

            cin >> val;

            node \*root = tree.insert(val);

            cout << "Balance factor of the tree after rotations  is now " << tree.balancefactor(root) << endl;

        }

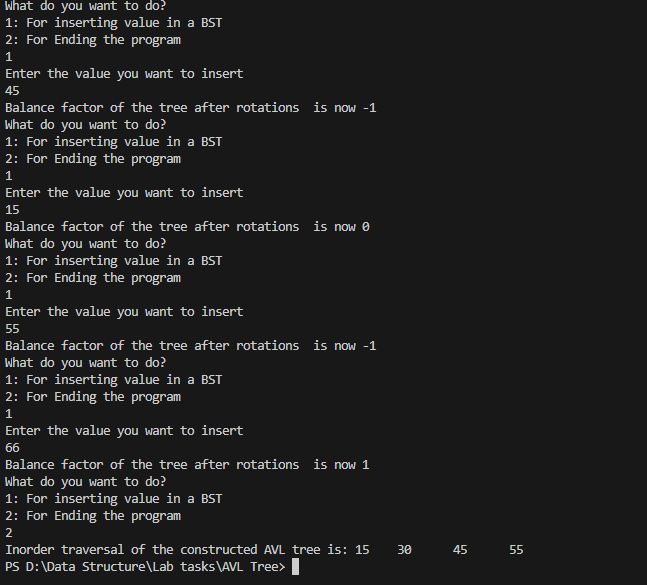
    } while (option != 2);

    cout << "Inorder traversal of the constructed AVL tree is: ";

    tree.display(tree.root);

    return 0;

}

OUTPUT

CODE 1:

#include <iostream>

using namespace std;

class node

{

public:

    int data;

    node \*left, \*right;

    node()

    {

        data = 0;

        left = NULL;

        right = NULL;

    }

    node(int data)

    {

        this->data = data;

        left = NULL;

        right = NULL;

    }

};

class BST

{

public:

    node \*root, \*n1;

    BST()

    {

        root = NULL;

    }

    void insert(int data)

    {

        n1 = new node(data);

        if (root == NULL)

        {

            root = n1;

            return;

        }

        else

        {

            node \*current = root;

            while (true)

            {

                if (data > current->data)

                {

                    if (current->right == NULL)

                    {

                        current->right = n1;

                        break;

                    }

                    current = current->right;

                }

                else

                {

                    if (current->left == NULL)

                    {

                        current->left = n1;

                        break;

                    }

                    current = current->left;

                }

            }

        }

    }

    int height(node \*r)

    {

        if (r == NULL)

        {

            return 0;

        }

        int lheight = height(r->left);

        int rheight = height(r->right);

        if (lheight > rheight)

            return (lheight + 1);

        else

            return (rheight + 1);

    }

    bool AVLORNOT(node \*ro)

    {

        int lh, rh;

        if (ro == NULL)

        {

            return 1;

        }

        lh = height(ro->left);

        rh = height(ro->right);

        int result = lh - rh;

        cout << "result is " << result<<endl;

        if (result == -1 || result == 0 || result == 1)

        {

            return true;

        }

        return false;

    }

    ~BST()

    {

        delete n1;

    }

};

int main()

{

    int option;

    BST tree;

    do

    {

        cout << "What do you want to do?\n";

        cout << "1: For inserting value in a BST\n";

        cout << "2: For checking a BST or not\n";

        cout << "3: For Ending the program\n";

        cin >> option;

        if (option == 1)

        {

            int val;

            cout << "Enter the value you want to insert\n";

            cin >> val;

            tree.insert(val);

        }

        else if (option == 2)

        {

            if (tree.root == NULL)

            {

                cout << "Tree is not created yet\n";

            }

            else

            {

                if (tree.AVLORNOT(tree.root))

                {

                    cout << "AVL";

                    break;

                }

                else

                {

                    cout << "NOT AVL";

                    break;

                }

            }

        }

    } while (option != 3);

    return 0;

}

OUTPUT:

