Lab Task  
1. Write a program in C++ to insert node in AVL Tree and maintain the balance factor.

#include<iostream>

using namespace std;

class node

{

    public:

    int height;

    int key;

    node \*left;

    node \*right;

};

int max(int a, int b)

{

    if(a>b){

        return a;

    }

    else if(b>a){

        return b;

    }

}

int height(node \*N)

{

    if (N == NULL)

        return 0;

    return N->height;

}

node\* newNode(int key)

{

    node\* new\_node = new node();

    new\_node->key = key;

    new\_node->left = NULL;

    new\_node->right = NULL;

    new\_node->height = 1;

    return new\_node;

}

node \*right\_Rotate(node \*y)

{

    node \*x = y->left;

    node \*z = x->right;

    x->right = y;

    y->left = z;

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

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

    return x;

}

node \*left\_Rotate(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 get\_Balance(node \*N)

{

    if (N == NULL)

        return 0;

    return height(N->left) - height(N->right);

}

node\* insert\_node(node\* new\_node, int key)

{

    if(new\_node == NULL){

        return(newNode(key));

    }

    else{

        if(key<new\_node->key){

            new\_node->left = insert\_node(new\_node->left, key);

        }

        else if(key>new\_node->key){

            new\_node->right = insert\_node(new\_node->right, key);

        }

        return new\_node;

    }

    new\_node->height = 1 + max(height(new\_node->left), height(new\_node->right));

    int balance = get\_Balance(new\_node);

    // LL

    if (balance > 1 && key < new\_node->left->key)

        return right\_Rotate(new\_node);

    // RR

    if (balance < -1 && key > new\_node->right->key)

        return left\_Rotate(new\_node);

    // LR

    if (balance > 1 && key > new\_node->left->key)

    {

        new\_node->left = left\_Rotate(new\_node->left);

        return right\_Rotate(new\_node);

    }

    // RL

    if (balance < -1 && key < new\_node->right->key)

    {

        new\_node->right = right\_Rotate(new\_node->right);

        return left\_Rotate(new\_node);

    }

    return new\_node;

}

void display(node\* current\_node) {

    if (current\_node != NULL) {

        cout << current\_node->key << endl;

        display(current\_node->left);

        display(current\_node->right);

    }

}

int main()

{

    node \*root = NULL;

    root = insert\_node(root, 10);

    root = insert\_node(root, 20);

    root = insert\_node(root, 30);

    root = insert\_node(root, 40);

    root = insert\_node(root, 50);

    int new\_node\_data;

    cout << "Enter the data : ";

    cin >> new\_node\_data;

    cout << endl;

    root = insert\_node(root, new\_node\_data);

    display(root);

    return 0;

}

