**Data Structure and Algorithm(Lab)**

**Name: Zeeshan Ali (SU92-BSITM-F22-019)**

**Lab Assignment**

**Question:**

**Code :**

#include <iostream>

using namespace std;

class AVL {

int data;

AVL \*left, \*right;

int height;

public:

AVL();

AVL(int);

AVL\* Insert(AVL\*, int);

AVL\* Delete(AVL\*, int);

void Inorder(AVL\*);

void preorder(AVL\*);

void postorder(AVL\*);

int getHeight(AVL\* node);

int getBalance(AVL\* node);

AVL\* rightRotate(AVL\* y);

AVL\* leftRotate(AVL\* x);

AVL\* minValueNode(AVL\* node);

};

AVL::AVL() {

data = 0;

left = right = NULL;

height = 1;

}

AVL::AVL(int value) {

data = value;

left = right = NULL;

height = 1;

}

AVL\* AVL::Insert(AVL\* root, int value) {

if (root == NULL) {

return new AVL(value);

}

if (value < root->data) {

root->left = Insert(root->left, value);

} else if (value > root->data) {

root->right = Insert(root->right, value);

}

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

int balance = getBalance(root);

if (balance > 1 && value < root->left->data) {

return rightRotate(root);

}

if (balance < -1 && value > root->right->data) {

return leftRotate(root);

}

if (balance > 1 && value > root->left->data) {

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

return rightRotate(root);

}

if (balance < -1 && value < root->right->data) {

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

return leftRotate(root);

}

return root;

}

int AVL::getHeight(AVL\* node) {

if (node == NULL) {

return 0;

}

return node->height;

}

int AVL::getBalance(AVL\* node) {

if (node == NULL) {

return 0;

}

return getHeight(node->left) - getHeight(node->right);

}

AVL\* AVL::rightRotate(AVL\* y) {

AVL\* x = y->left;

AVL\* T2 = x->right;

x->right = y;

y->left = T2;

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

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

return x;

}

AVL\* AVL::leftRotate(AVL\* x) {

AVL\* y = x->right;

AVL\* T2 = y->left;

y->left = x;

x->right = T2;

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

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

return y;

}

AVL\* AVL::minValueNode(AVL\* node) {

AVL\* current = node;

while (current->left != NULL) {

current = current->left;

}

return current;

}

void AVL::Inorder(AVL\* root) {

if (root == NULL) {

return;

}

Inorder(root->left);

cout << root->data << " ";

Inorder(root->right);

}

void AVL::preorder(AVL\* root) {

if (root == NULL) {

return;

}

cout << root->data << " ";

preorder(root->left);

preorder(root->right);

}

void AVL::postorder(AVL\* root) {

if (root == NULL) {

return;

}

postorder(root->left);

postorder(root->right);

cout << root->data << " ";

}

int main() {

AVL avl, \*root = NULL;

root = avl.Insert(root, 10);

root = avl.Insert(root, 5);

root = avl.Insert(root, 15);

root = avl.Insert(root, 3);

root = avl.Insert(root, 7);

root = avl.Insert(root, 12);

root = avl.Insert(root, 18);

root = avl.Insert(root, 2);

root = avl.Insert(root, 4);

root = avl.Insert(root, 6);

root = avl.Insert(root, 8);

root = avl.Insert(root, 11);

root = avl.Insert(root, 14);

root = avl.Insert(root, 13);

root = avl.Insert(root, 19);

root = avl.Insert(root, 20);

cout << "Inorder traversal" << endl;

avl.Inorder(root);

cout << "\nPreorder traversal" << endl;

avl.preorder(root);

cout << "\nPostorder traversal" << endl;

avl.postorder(root);

return 0;

}

**Output:**

