Ans.1 –

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

int height;

};

class AVLTree {

public:

Node\* root;

int height(Node\* node) {

if (node == nullptr)

return 0;

return node->height;

}

int getBalanceFactor(Node\* node) {

if (node == nullptr)

return 0;

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

}

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;

}

Node\* insert(Node\* node, int key) {

if (node == nullptr) {

Node\* newNode = new Node();

newNode->data = key;

newNode->left = nullptr;   
 newNode->right = nullptr

newNode->height = 1;

return newNode;

}

if (key < node->data)

node->left = insert(node-

>left, key);

else if (key > node->data)

node->right = insert(node->right, key);

else

return node;

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

int balance = getBalanceFactor(node);

if (balance > 1) {

if (key < node->left->data) {

return rightRotate(node);

} else {

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

return rightRotate(node);

}

}

if (balance < -1) {

if (key > node->right>data) {

return leftRotate(node);

} else {

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

return leftRotate(node);

}

}

return node;

}

void insert(int key) {

root = insert(root, key);

}

void printTree(Node\* node) {

if (node == nullptr) return; printTree(node->left);

cout << "Data: " << node->data << " Height: " << node->height << " Balance Factor: " << getBalanceFactor(node) << endl;

printTree(node->right);

}

void printTree() {

printTree(root);

}

};

int main() {

AVLTree tree;

int numbers[] = {10, 5, 15, 3, 7, 12, 18}; int numCount =

sizeof(numbers) / sizeof(numbers[0]);

for (int i = 0; i < numCount; i++) {

tree.insert(numbers[i]);

cout << "After inserting " << numbers[i] << ":" << endl;

tree.printTree();

cout << "-----------------------------" << endl;

}

return 0;

}

Ans. 2 –

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

int height;

};

class AVLTree {

public: Node\* root;

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;

}

Node\* leftLeftRotate(Node\* node) {

return rightRotate(node);

}

Node\* leftRightRotate(Node\* node) {

node->left = leftRotate(node->left); return rightRotate(node);

}

Node\* rightLeftRotate(Node\* node) {

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

return leftRotate(node);

}

Node\* rightRightRotate(Node\* node) {

return leftRotate(node);

}

};

Ans. 3 –

a. To insert 16 into the tree, we need to perform rotations to maintain the AVL tree's balance. In this case, a left-right rotation is required.

After inserting 16, the tree becomes balanced.

10

/ \

5 15

/ \ / \

3 7 12 18

/

16

b. Deleting the node with the value 10 requires some rotations. After deleting 10, the tree will become balanced. The resulting tree looks like this:

12

/ \

5 15

/ \ \

3 7 18

/

16

1. To search for a node with the value 7, you find it in the tree. The height of this node is 2.
2. InOrder Traversal of the tree will give you the values in ascending order:

3 -> 5 -> 7 -> 12 -> 15 -> 16 -> 18

e. PreOrder Traversal of the tree: 12 -> 5 > 3 -> 7 -> 15 -> 18 -> 16 f. PostOrder Traversal of the tree: 3 -> 7 -> 5 -> 16 -> 18 -> 15 -> 12

g. You can delete the entire AVL tree by recursively deleting the nodes starting from the leaves

and moving up to the root. Here's the tree after deleting all nodes: (Empty Tree)