QUESTION 1:

INSERTION FUNTION AND IMPLEMENTATION OF AVL TREES

#include <stdio.h>

#include <stdlib.h>

struct Node {

int key;

struct Node\* left;

struct Node\* right;

int height;

};

int max(int a, int b) {

return (a > b) ? a : b;

}

int height(struct Node\* node) {

if (node == NULL) return 0;

return node->height;

}

int balance\_factor(struct Node\* node) {

if (node == NULL) return 0;

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

}

struct Node\* new\_node(int key) {

struct Node\* node = (struct Node\*)malloc(sizeof(struct Node));

node->key = key;

node->left = NULL;

node->right = NULL;

node->height = 1;

return node;

}

struct Node\* rotate\_left(struct Node\* y) {

struct Node\* x = y->right;

struct Node\* T2 = x->left;

x->left = y;

y->right = T2;

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

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

return x;

}

struct Node\* rotate\_right(struct Node\* x) {

struct Node\* y = x->left;

struct Node\* T2 = y->right;

y->right = x;

x->left = T2;

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

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

return y;

}

struct Node\* insert\_node(struct Node\* root, int key) {

if (root == NULL) return new\_node(key);

if (key < root->key) {

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

} else {

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

}

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

int balance = balance\_factor(root);

// Left heavy

if (balance > 1) {

if (key < root->left->key) {

printf("Right Rotation on node with key %d\n", root->key);

return rotate\_right(root);

} else {

printf("Left-Right Rotation on node with key %d\n", root->key);

root->left = rotate\_left(root->left);

return rotate\_right(root);

}

}

// Right heavy

if (balance < -1) {

if (key > root->right->key) {

printf("Left Rotation on node with key %d\n", root->key);

return rotate\_left(root);

} else {

printf("Right-Left Rotation on node with key %d\n", root->key);

root->right = rotate\_right(root->right);

return rotate\_left(root);

}

}

return root;

}

void print\_balance\_factors(struct Node\* root) {

if (root != NULL) {

printf("Balance factor of node with key %d is %d\n", root->key, balance\_factor(root));

print\_balance\_factors(root->left);

print\_balance\_factors(root->right);

}

}

int main() {

struct Node\* root = NULL;

int k = 6;

int keys[] = {10, 20, 30, 40, 50, 25};

int i;

for ( i = 0; i < k; i++) {

root = insert\_node(root, keys[i]);

printf("Inserted node with key %d\n", keys[i]);

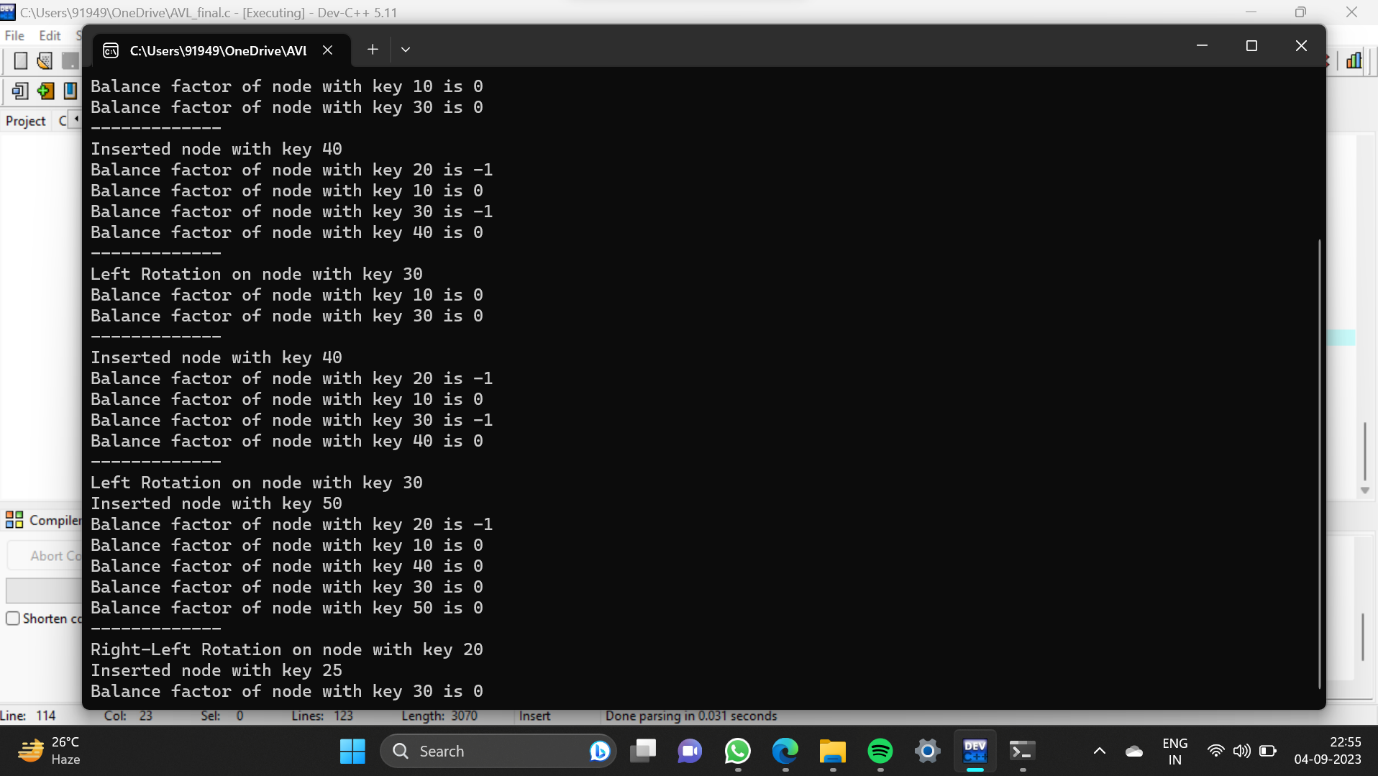
print\_balance\_factors(root);

printf("-------------\n");

}

return 0;

}



QUESTION 2:

IMPLEMENTATION OF DELETION OPERATION

struct Node \*deleteNode(struct Node \*root, int key) {

// Find the node and delete it

if (root == NULL)

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 == NULL) || (root->right == NULL)) {

struct Node \*temp = root->left ? root->left : root->right;

if (temp == NULL) {

temp = root;

root = NULL;

} else

\*root = \*temp;

free(temp);

} else {

struct Node \*temp = minValueNode(root->right);

root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

}

if (root == NULL)

return root;

// Update the balance factor of each node and

// balance the tree

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)

return leftRotate(root);

if (balance < -1 && getBalance(root->right) > 0) {

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

return leftRotate(root);

}

return root;

}

//printing the tree in the pre-order traversal mode

void printPreOrder(struct Node \*root) {

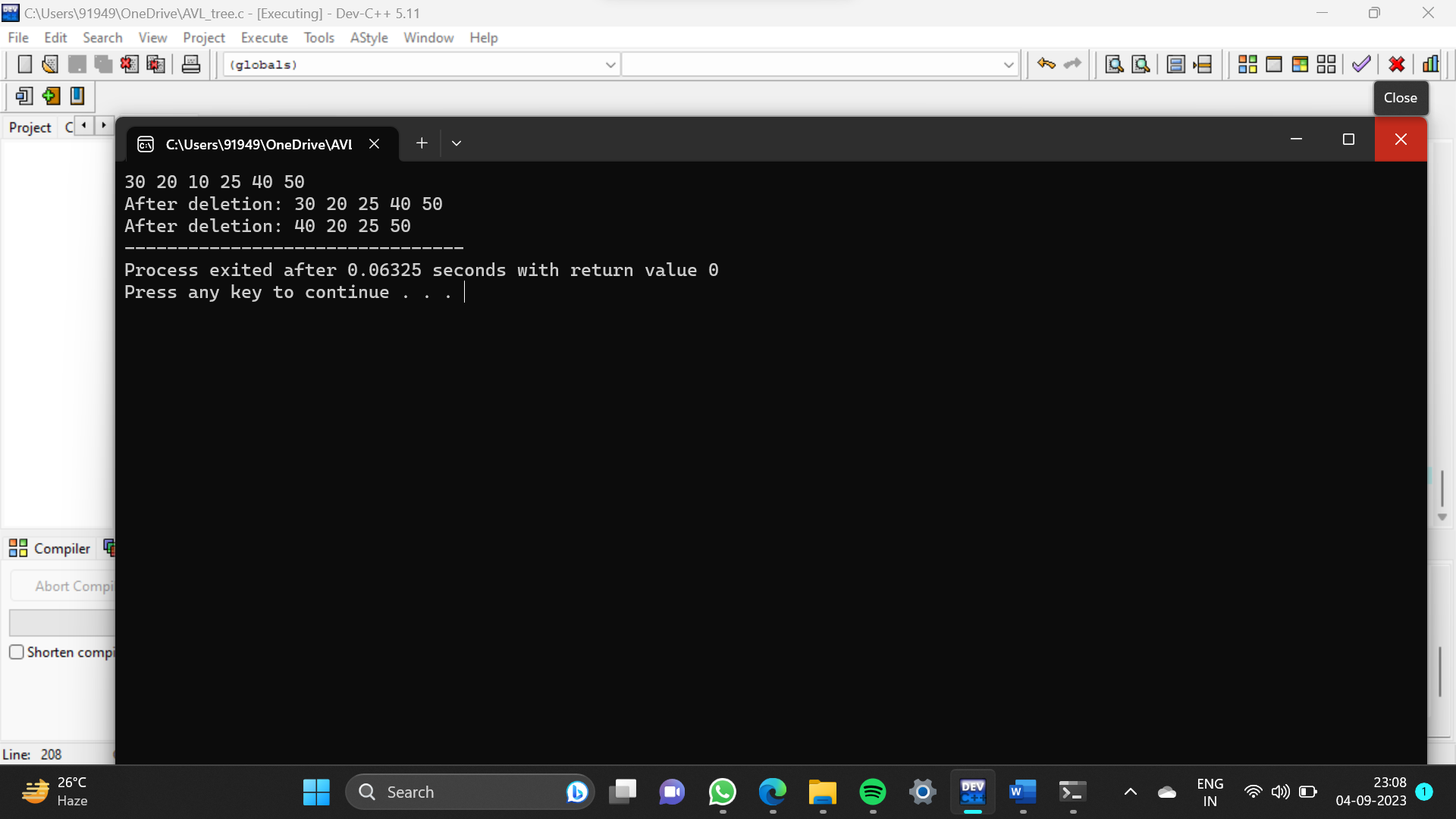
if (root != NULL) {

printf("%d ", root->key);

printPreOrder(root->left);

printPreOrder(root->right);

}

} 

Question 3:

int isAVLTree(struct TreeNode\* root) {

if (root == NULL)

return 1;

int left\_height = height(root-&gt;left);

int right\_height = height(root-&gt;right);

if (abs(left\_height - right\_height) &gt; 1)

return 0;

return isAVLTree(root-&gt;left) &amp;&amp; isAVLTree(root-&gt;right);

}

