Practical 9

Aim: Write a program to perform AVL insertion and deletion.

Code:

```
#include <iostream>
using namespace std;
struct Node {
   int key;
   Node* left;
   Node* right;
   int height;
};
int height(Node* N) {
   return (N == nullptr) ? 0 : N->height;
}
int max(int a, int b) {
   return (a > b) ? a : b;
}
Node* createNode(int key) {
   Node* node = new Node();
   node->key = key;
   node->left = node->right = nullptr;
   node->height = 1;
   return node;
}
Node* rightRotate(Node* y) {
   Node* x = y->left;
   Node* T2 = x->right;

   x->right = y;
```

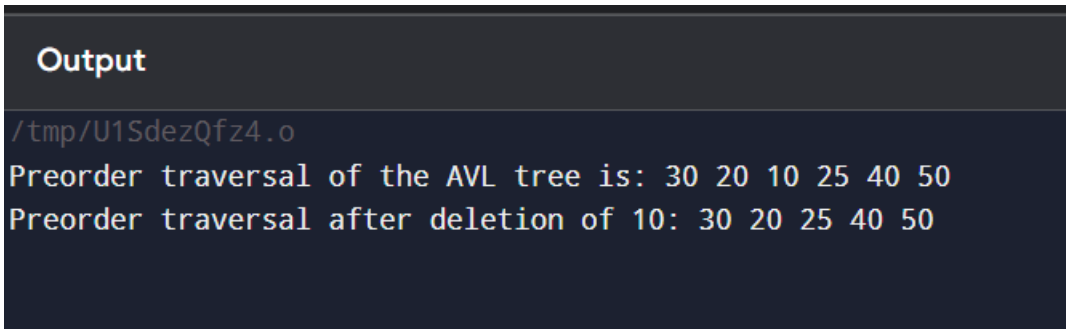
```
    y->left = T2;

    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;
}
int getBalance(Node* N) {
    return (N == nullptr) ? 0 : height(N->left) - height(N->right);
}
Node* insert(Node* node, int key) {
    if (node == nullptr)
        return createNode(key);
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    else
        return node;
```

```
    node->height = 1 + max(height(node->left), height(node->right));
    int balance = getBalance(node);
    if (balance > 1 && key < node->left->key)
        return rightRotate(node);
    if (balance < -1 && key > node->right->key)
        return leftRotate(node);
    if (balance > 1 && key > node->left->key) {
        node->left = leftRotate(node->left);
        return rightRotate(node);
    }
    if (balance < -1 && key < node->right->key) {
        node->right = rightRotate(node->right);
        return leftRotate(node);
    }
    return node;
}
Node* minValueNode(Node* node) {
    Node* current = node;
    while (current->left != nullptr)
        current = current->left;
    return current;
}
Node* deleteNode(Node* root, int key) {
    if (root == nullptr)
        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 == nullptr) || (root->right == nullptr)) {
        Node* temp = root->left ? root->left : root->right;
        if (temp == nullptr) {
            temp = root;
            root = nullptr;
        } else
            *root = *temp;
        delete temp;
    } else {
        Node* temp = minValueNode(root->right);
        root->key = temp->key;
        root->right = deleteNode(root->right, temp->key);
    }
}

if (root == nullptr)
    return root;
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;
}

void preOrder(Node* root) {
    if (root != nullptr) {
        cout << root->key << " ";
        preOrder(root->left);
        preOrder(root->right);
    }
}
int main() {
    Node* root = nullptr;
    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 40);
    root = insert(root, 50);
    root = insert(root, 25);

    cout << "Preorder traversal of the AVL tree is: ";
    preOrder(root);
    cout << endl;
    root = deleteNode(root, 10);
    cout << "Preorder traversal after deletion of 10: ";
    preOrder(root);
    cout << endl;

    return 0;
}
```

Output:


Output
/tmp/U1SdezQfz4.o
Preorder traversal of the AVL tree is: 30 20 10 25 40 50
Preorder traversal after deletion of 10: 30 20 25 40 50