#include <iostream>

#include <cstdlib> // For exit() function

using namespace std;

#define FALSE 0

#define TRUE 1

struct AVLNode {

int data;

int balfact;

AVLNode \*left;

AVLNode \*right;

};

// Function prototypes

AVLNode\* buildtree(AVLNode\*, int, int\*);

AVLNode\* deldata(AVLNode\*, int, int\*);

AVLNode\* del(AVLNode\*, AVLNode\*, int\*);

AVLNode\* balright(AVLNode\*, int\*);

AVLNode\* balleft(AVLNode\*, int\*);

void display(AVLNode\*);

void deltree(AVLNode\*);

int main() {

AVLNode\* avl = nullptr;

int h, val, ch;

while (true) {

cout << "\n1: Insert\n2: Delete\n3: Display\n4: Delete Tree\n5: Exit\n";

cin >> ch;

switch (ch) {

case 1:

cout << "\nEnter the data: ";

cin >> val;

avl = buildtree(avl, val, &h);

break;

case 2:

cout << "\nEnter the data: ";

cin >> val;

avl = deldata(avl, val, &h);

break;

case 3:

cout << "\nAVL tree (in-order traversal):\n";

display(avl);

cout << endl;

break;

case 4:

deltree(avl);

avl = nullptr;

cout << "\nTree deleted.";

break;

case 5:

deltree(avl);

exit(0);

default:

cout << "\nInvalid choice. Try again.";

}

}

}

// Function to insert an element into the AVL tree

AVLNode\* buildtree(AVLNode\* root, int data, int\* h) {

if (!root) {

root = new AVLNode;

root->data = data;

root->left = nullptr;

root->right = nullptr;

root->balfact = 0;

\*h = TRUE;

return root;

}

if (data < root->data) {

root->left = buildtree(root->left, data, h);

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

root->right = buildtree(root->right, data, h);

}

return root;

}

// Function to delete a node from the AVL tree

AVLNode\* deldata(AVLNode\* root, int data, int\* h) {

if (!root) {

cout << "\nNo such data.";

return root;

}

if (data < root->data) {

root->left = deldata(root->left, data, h);

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

root->right = deldata(root->right, data, h);

} else {

AVLNode\* temp = nullptr;

if (!root->left) {

temp = root->right;

delete root;

return temp;

} else if (!root->right) {

temp = root->left;

delete root;

return temp;

} else {

AVLNode\* successor = root->right;

while (successor->left) successor = successor->left;

root->data = successor->data;

root->right = deldata(root->right, successor->data, h);

}

}

return root;

}

// Function to balance the tree if the right subtree is higher

AVLNode\* balright(AVLNode\* root, int\* h) {

return root; // Placeholder (Not Implemented)

}

// Function to balance the tree if the left subtree is higher

AVLNode\* balleft(AVLNode\* root, int\* h) {

return root; // Placeholder (Not Implemented)

}

// Function to display the tree (in-order traversal)

void display(AVLNode\* root) {

if (root) {

display(root->left);

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

display(root->right);

}

}

// Function to delete the tree

void deltree(AVLNode\* root) {

if (root) {

deltree(root->left);

deltree(root->right);

delete root;

}

}

  