Assignment No.4

Problem Statement :

Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree -

i. Insert new node
ii. Find number of nodes in longest path from root
iii. Minimum data value found in the tree
iv. Change a tree so that the roles of the left and right pointers are swapped at every node
v. Search a value

Program :

```
#include <iostream>

#include <algorithm>


using namespace std;


// Node structure

struct Node {

   int data;

   Node* left;

   Node* right;


   Node(int val) : data(val), left(nullptr), right(nullptr) {}

};


// Function to insert a new node in the BST

Node* insert(Node* root, int val) {

   if (root == nullptr) {

      return new Node(val);
```

```
    }

    if (val < root->data) {
        root->left = insert(root->left, val);
    } else {
        root->right = insert(root->right, val);
    }

    return root;
}

// Function to find the number of nodes in the longest path from root
int longestPath(Node* root) {
    if (root == nullptr) {
        return 0;
    }

    int leftHeight = longestPath(root->left);
    int rightHeight = longestPath(root->right);

    return max(leftHeight, rightHeight) + 1;  // Height of the tree (longest path from root)
}

// Function to find the minimum value in the BST
int findMin(Node* root) {
    if (root == nullptr) {
        cout << "Tree is empty!" << endl;
        return -1;
    }

    while (root->left != nullptr) {
```

```
        root = root->left;
    }
    return root->data;
}

// Function to swap left and right pointers at every node
void swapSubtrees(Node* root) {
    if (root == nullptr) {
        return;
    }

    // Swap left and right children
    swap(root->left, root->right);

    // Recur for left and right subtrees
    swapSubtrees(root->left);
    swapSubtrees(root->right);
}

// Function to search for a value in the BST
bool search(Node* root, int val) {
    if (root == nullptr) {
        return false;
    }

    if (root->data == val) {
        return true;
    } else if (val < root->data) {
        return search(root->left, val);
    } else {
        return search(root->right, val);
```

```
    }
}

// Function to print the tree in-order
void inorder(Node* root) {
    if (root == nullptr) return;
    inorder(root->left);
    cout << root->data << " ";
    inorder(root->right);
}

int main() {
    Node* root = nullptr;

    // Constructing binary search tree with given values
    int values[] = {50, 30, 20, 40, 70, 60, 80};
    for (int val : values) {
        root = insert(root, val);
    }

    cout << "In-order traversal of BST before operations: ";
    inorder(root);
    cout << endl;

    // i. Insert a new node with value 25
    root = insert(root, 25);
    cout << "In-order traversal after inserting 25: ";
    inorder(root);
    cout << endl;

    // ii. Find the number of nodes in the longest path (height)
```

```
    int longestPathLength = longestPath(root);
    cout << "Longest path from root: " << longestPathLength << endl;

    // iii. Find minimum data value in the tree
    int minValue = findMin(root);
    cout << "Minimum value in the tree: " << minValue << endl;

    // iv. Swap left and right pointers at each node
    swapSubtrees(root);
    cout << "In-order traversal after swapping left and right children: ";
    inorder(root);
    cout << endl;

    // v. Search for value 40
    int searchValue = 40;
    bool found = search(root, searchValue);
    if (found) {
        cout << "Value " << searchValue << " found in the tree." << endl;
    } else {
        cout << "Value " << searchValue << " not found in the tree." << endl;
    }

    return 0;
}
```

Output :

Output

Clear

```
In-order traversal of BST before operations: 20 30 40 50 60 70 80
In-order traversal after inserting 25: 20 25 30 40 50 60 70 80
Longest path from root: 4
Minimum value in the tree: 20
In-order traversal after swapping : 80 70 60 50 40 30 25 20
Value 40 not found in the tree.


=== Code Execution Successful ===
```