Binary Search Tree (BST) Implementation:

Create a Node structure with data, left, and right pointers.

Implement functions for:

Insertion: *Recursively add nodes while maintaining the BST property (left subtree < node < right subtree).*

Deletion: *Handle different cases (leaf node, single child, two children).*

Search: *Recursively search for a specific value, returning the node or NULL if not found.*

In-order Traversal: *Print nodes in ascending order (left subtree, node, right subtree).*

Pre-order and Post-order Traversal of a Binary Tree:

Modify the in-order traversal function from question 1 to perform pre-order (root, left, right) and post-order (left, right, root) traversals.

 Check if a Binary Tree is a Binary Search Tree (BST):

Implement a recursive function that checks if the left subtree is less than the current node and the right subtree is greater than the current node.

Traverse the tree and return false if any violation is found, otherwise return true.

Find the Height of a Binary Tree:

Implement a recursive function that returns the maximum depth (height) from the root to a leaf node.

Find the Minimum and Maximum Values in a Binary Tree:

Modify the in-order traversal function to track the minimum and maximum values encountered so far.

bstapi.c

```c
#include "bstfunc.h"

#include <stdio.h>

#include <stdlib.h>


TreeNode* createNode(int data) {

    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));

    if (newNode == NULL) {

        printf("Memory allocation failed.\n");

        exit(EXIT_FAILURE);

    }

    newNode->data = data;

    newNode->left = NULL;

    newNode->right = NULL;

    return newNode;

}


TreeNode* insert(TreeNode* root, int data) {

    if (root == NULL)

        return createNode(data);

    if (data < root->data)

        root->left = insert(root->left, data);

    else if (data > root->data)

        root->right = insert(root->right, data);

    return root;

}


TreeNode* minValueNode(TreeNode* node) {

    TreeNode* current = node;
```

```c
    while (current && current->left != NULL)

        current = current->left;

    return current;

}


TreeNode* deleteNode(TreeNode* root, int key) {

    if (root == NULL)

        return root;

    if (key < root->data)

        root->left = deleteNode(root->left, key);

    else if (key > root->data)

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

    else {

        if (root->left == NULL) {

            TreeNode* temp = root->right;

            free(root);

            return temp;

        }

        else if (root->right == NULL) {

            TreeNode* temp = root->left;

            free(root);

            return temp;

        }

        TreeNode* temp = minValueNode(root->right);

        root->data = temp->data;

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

    }

    return root;

}


TreeNode* search(TreeNode* root, int key) {
```

```c
    if (root == NULL || root->data == key)
        return root;
    if (root->data < key)
        return search(root->right, key);
    return search(root->left, key);
}


void inorderTraversal(TreeNode* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}


void preorderTraversal(TreeNode* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}


void postorderTraversal(TreeNode* root) {
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}
```

```c
int isBSTUtil(TreeNode* root, int min_val, int max_val) {

    if (root == NULL)

        return 1;

    if (root->data < min_val || root->data > max_val)

        return 0;

    return isBSTUtil(root->left, min_val, root->data - 1) && isBSTUtil(root->right, root->data + 1,
max_val);

}


int isBST(TreeNode* root) {

    return isBSTUtil(root, INT_MIN, INT_MAX);

}


int max(int a, int b) {

    return (a > b) ? a : b;

}


int findHeight(TreeNode* root) {

    if (root == NULL)

        return 0;

    int leftHeight = findHeight(root->left);

    int rightHeight = findHeight(root->right);

    return 1 + max(leftHeight, rightHeight);

}


void findMinMax(TreeNode* root, int* min, int* max) {

    if (root == NULL)

        return;

    if (root->data < *min)

        *min = root->data;

    if (root->data > *max)
```

```c
        *max = root->data;

    findMinMax(root->left, min, max);

    findMinMax(root->right, min, max);
}
```

binarysearchtree.h

```c
#ifndef BSTFUNC_H
#define BSTFUNC_H


// Structure for a Node in BST
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};


typedef struct TreeNode TreeNode;


// Function declarations
TreeNode* createNode(int data);
TreeNode* insert(TreeNode* root, int data);
TreeNode* deleteNode(TreeNode* root, int key);
TreeNode* search(TreeNode* root, int key);
void inorderTraversal(TreeNode* root);
void preorderTraversal(TreeNode* root);
void postorderTraversal(TreeNode* root);
int isBST(TreeNode* root);
int findHeight(TreeNode* root);
void findMinMax(TreeNode* root, int* min, int* max);
```

```c
#endif /* BSTFUNC_H */
```

bstmain.c

```c
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include "bstfunc.h"


int main() {

    clock_t start, end;

    double cpu_time_used;


    // Creating a sample BST

    TreeNode* root = NULL;

    root = insert(root, 4);

    insert(root, 2);

    insert(root, 6);

    insert(root, 1);

    insert(root, 3);

    insert(root, 5);

    insert(root, 7);


    // Insertion and deletion

    root = insert(root, 8);

    root = deleteNode(root, 3);


    // Search

    int searchKey = 5;

    TreeNode* searchedNode = search(root, searchKey);

    if (searchedNode != NULL)
```

```c
        printf("%d found in the BST.\n", searchKey);
    else
        printf("%d not found in the BST.\n", searchKey);


    // In-order traversal
    printf("In-order traversal: ");
    inorderTraversal(root);
    printf("\n");


    // Pre-order traversal
    printf("Pre-order traversal: ");
    preorderTraversal(root);
    printf("\n");


    // Post-order traversal
    printf("Post-order traversal: ");
    postorderTraversal(root);
    printf("\n");


    // Check if BST
    if (isBST(root))
        printf("The tree is a Binary Search Tree.\n");
    else
        printf("The tree is not a Binary Search Tree.\n");


    // Height of the tree
    int height = findHeight(root);
    printf("Height of the tree is: %d\n", height);


    // Min and Max values
    int min = INT_MAX, max = INT_MIN;
```

```
findMinMax(root, &min, &max);

printf("Minimum value in the tree: %d\n", min);

printf("Maximum value in the tree: %d\n", max);


// Measure time taken

start = clock();

// Perform operations here...

end = clock();

cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

printf("Time taken: %f seconds\n", cpu_time_used);


return 0;
}
```

Explanation

Node Structure:

A node in a BST contains three parts: data, left pointer, and right pointer.

The data represents the value stored in the node.

The left pointer points to the left child node.

The right pointer points to the right child node.

Insertion:

To insert a value into a BST, start from the root node.

If the tree is empty, create a new node as the root with the given value.

Otherwise,

Recursively traverse the tree:

If the value is less than the current node's data, move to the left subtree.

If the value is greater than the current node's data, move to the right subtree.

Repeat until an appropriate empty spot is found, then insert the new node.

Deletion:

Deleting a node from a BST requires handling different cases:

If the node to delete is a leaf node, simply remove it.

If the node has only one child, bypass it by connecting its parent directly to its child.

If the node has two children, find its successor (the minimum value in its right subtree), replace the node's data with the successor's data, and recursively delete the successor node.

Search:

Searching for a value in a BST involves recursively traversing the tree:

If the current node is NULL or contains the value, return the node.

If the value is less than the current node's data, search the left subtree.

If the value is greater than the current node's data, search the right subtree.

In-order Traversal:

In-order traversal prints the nodes of the BST in ascending order:

Traverse the left subtree recursively.

Visit the current node.

Traverse the right subtree recursively.

Pre-order and Post-order Traversal of a Binary Tree:

Pre-order traversal starts from the root, then visits the left subtree, and finally visits the right subtree.

Post-order traversal starts from the left subtree, then visits the right subtree, and finally visits the root.

Check if a Binary Tree is a Binary Search Tree (BST):

The function recursively checks if the left subtree is less than the current node and the right subtree is greater than the current node.

It traverses the tree and returns false if any violation is found, otherwise returns true.

Find the Height of a Binary Tree:

The function returns the maximum depth (height) from the root to a leaf node.

It recursively calculates the height of the left and right subtrees, then returns the maximum height plus one.

Find the Minimum and Maximum Values in a Binary Tree:

Modify the in-order traversal function to track the minimum and maximum values encountered so far.

At each node, update the minimum and maximum values if necessary.

output:

```
rps@rps-virtual-machine:~/LDD_Batch/Sarath/project_bst$ ./makebst
BINARY TREE USING INORDER TRAVERSING :20 30 40 50 60 70 80
BINARY TREE USING PRE :50 30 20 40 70 60 80
BINARY TREE USING POSTORDER :20 40 30 60 80 70 50
Height of tree is : 3
NUMBER OF NODES : 7
Enter element to find:20
Yes,20 is found
After Deleting Required Node : 20 30 40 50 60 80
Minimum element in Tree : 20
Maximum element in Tree: 80
Sum of elements is 280
Time taken is 333.000000
rps@rps-virtual-machine:~/LDD_Batch/Sarath/project_bst$ gcc -O2 bstmain.c bstapi.c -o bstO2
bstmain.c: In function 'main':
bstmain.c:43:9: warning: ignoring return value of 'scanf' declared with attribute 'warn_unused_result' [-Wunused-result]
   43 |         scanf("%d",&ele); //TAKING FROM USER WHICH ELEMENT TO FIND
      |         ^~~~~~~~~~~~~~~~
rps@rps-virtual-machine:~/LDD_Batch/Sarath/project_bst$ ./bstO2
BINARY TREE USING INORDER TRAVERSING :20 30 40 50 60 70 80
BINARY TREE USING PRE :50 30 20 40 70 60 80
BINARY TREE USING POSTORDER :20 40 30 60 80 70 50
Height of tree is : 3
NUMBER OF NODES : 7
Enter element to find:30
Yes,30 is found
After Deleting Required Node : 20 30 40 50 60 80
Minimum element in Tree : 20
Maximum element in Tree: 80
Sum of elements is 280
Time taken is 461.000000
```

```
rps@rps-virtual-machine:~/LDD_Batch/Sarath/project_bst$ gcc -O3 bstmain.c bstapi.c -o bstO3
bstmain.c: In function 'main':
bstmain.c:43:9: warning: ignoring return value of 'scanf' declared with attribute 'warn_unused_result' [-Wunused-result]
   43 |         scanf("%d",&ele); //TAKING FROM USER WHICH ELEMENT TO FIND
      |         ^~~~~~~~~~~~~~~~
rps@rps-virtual-machine:~/LDD_Batch/Sarath/project_bst$ ./bstO3
BINARY TREE USING INORDER TRAVERSING :20 30 40 50 60 70 80
BINARY TREE USING PRE :50 30 20 40 70 60 80
BINARY TREE USING POSTORDER :20 40 30 60 80 70 50
Height of tree is : 3
NUMBER OF NODES : 7
Enter element to find:40
Yes,40 is found
After Deleting Required Node : 20 30 40 50 60 80
Minimum element in Tree : 20
Maximum element in Tree: 80
Sum of elements is 280
Time taken is 276.000000
rps@rps-virtual-machine:~/LDD_Batch/Sarath/project_bst$ gcc -O4 bstmain.c bstapi.c -o bstO4
bstmain.c: In function 'main':
bstmain.c:43:9: warning: ignoring return value of 'scanf' declared with attribute 'warn_unused_result' [-Wunused-result]
   43 |         scanf("%d",&ele); //TAKING FROM USER WHICH ELEMENT TO FIND
      |         ^~~~~~~~~~~~~~~~
rps@rps-virtual-machine:~/LDD_Batch/Sarath/project_bst$ ./bstO4
BINARY TREE USING INORDER TRAVERSING :20 30 40 50 60 70 80
BINARY TREE USING PRE :50 30 20 40 70 60 80
BINARY TREE USING POSTORDER :20 40 30 60 80 70 50
Height of tree is : 3
NUMBER OF NODES : 7
Enter element to find:50
Yes,50 is found
After Deleting Required Node : 20 30 40 50 60 80
Minimum element in Tree : 20
Maximum element in Tree: 80
Sum of elements is 280
Time taken is 279.000000
```

```
START
  │
  ▼
CREATE AN EMPTY ROOT NODE
  │
  ▼
INSERT NODES INTO THE BINARY TREE
  │
  ▼
PRINT "IN-ORDER TRAVERSAL"
  │
  ▼
PERFORM IN-ORDER TRAVERSAL
  │
  ▼
PRINT THE HEIGHT OF THE BINARY TREE
  │
  ▼
FIND MINIMUM AND MAXIMUM VALUES IN THE BINARY TREE
  │
  ▼
PRINT MINIMUM AND MAXIMUM VALUES
  │
  ▼
CHECK IF THE BINARY TREE IS A BST
  │
  ▼
PRINT WHETHER IT'S A BST OR NOT
  │
  ▼
DELETE A NODE FROM THE BINARY TREE
  │
  ▼
PRINT "IN-ODER TRAVERSAL AFTER DELETION
  │
  ▼
PERFORM IN-ORDER TRAVERSAL AFTER DELETION
  │
  ▼
FREE THE MEMORY ALLOCATED FOR THE BINARY TREE
  │
  ▼
END
```