Rajalakshmi Engineering College

Name: vinnarasan j
Email: 241501245@rajalakshmi.edu.in
Roll no: 241501245
Phone: 7305999373
Branch: REC
Department: I AI & ML FC
Batch: 2028
Degree: B.E - AI & ML

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 5_COD_Question 4

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

John, a computer science student, is learning about binary search trees (BST) and their properties. He decides to write a program to create a BST, display it in post-order traversal, and find the minimum value present in the tree.

Help him by implementing the program.

Input Format

The first line of input consists of an integer N, representing the number of elements to insert into the BST.

The second line consists of N space-separated integers data, which is the data to be inserted into the BST.

Output Format

The first line of output prints the space-separated elements of the BST in post-order traversal.

The second line prints the minimum value found in the BST.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 3
5 10 15
Output: 15 10 5
The minimum value in the BST is: 5

Answer

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

// Structure for a node in the binary search tree
struct TreeNode {
    int val;
    struct TreeNode *left;
```

```
    struct TreeNode *right;
};

// Function to create a new node
struct TreeNode* createNode(int val) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
    newNode->val = val;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Function to insert a value into the BST
struct TreeNode* insert(struct TreeNode* root, int val) {
    if (root == NULL) {
        return createNode(val);
    }
    if (val < root->val) {
        root->left = insert(root->left, val);
    } else {
        root->right = insert(root->right, val);
    }
    return root;
}

// Function to perform post-order traversal of the BST
void postOrderTraversal(struct TreeNode* root) {
    if (root != NULL) {
        postOrderTraversal(root->left);
        postOrderTraversal(root->right);
        printf("%d ", root->val);
    }
}

// Function to find the minimum value in the BST
int findMinValue(struct TreeNode* root) {
    if (root == NULL) {
        return INT_MAX; // Return maximum integer value for an empty tree
    }

    //Iterative approach to find min value.
```

```
    while(root->left){
        root = root->left;
    }
    return root->val;
}

// Function to free the memory allocated for the BST
void freeTree(struct TreeNode* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}

int main() {
    int n, val;
    struct TreeNode* root = NULL;

    // Read the number of elements
    scanf("%d", &n);

    // Read the elements and insert them into the BST
    for (int i = 0; i < n; i++) {
        scanf("%d", &val);
        root = insert(root, val);
    }

    // Print the post-order traversal of the BST
    postOrderTraversal(root);
    printf("\n");

    // Find and print the minimum value
    int minValue = findMinValue(root);
    printf("The minimum value in the BST is: %d\n", minValue);

    freeTree(root); // Free the allocated memory
    return 0;
}
int main() {
    struct Node* root = NULL;
    int n, data;
```

```
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        root = insert(root, data);
    }

    displayTreePostOrder(root);
    printf("\n");

    int minValue = findMinValue(root);
    printf("The minimum value in the BST is: %d", minValue);

    return 0;
}
```

Status : Correct **Marks : 10/10**