Rajalakshmi Engineering College

Name: Nitin Aakash
Email: 240701370@rajalakshmi.edu.in
Roll no: 240701370
Phone: 9498349045
Branch: REC
Department: I CSE FD
Batch: 2028
Degree: B.E - CSE

Scan to verify results

NeoColab_REC_CS23231_DATA STRUCTURES

REC_DS using C_Week 5_CY_Updated

Attempt : 1
Total Mark : 30
Marks Obtained : 30

Section 1 : Coding

1. Problem Statement

John is building a system to store and manage integers using a binary search tree (BST). He needs to add a feature that allows users to search for a specific integer key in the BST using recursion.

Implement functions to create the BST and perform a recursive search for an integer.

Input Format

The first line of input consists of an integer representing, the number of nodes.

The second line consists of integers representing, the values of nodes, separated by space.

The third line consists of an integer representing, the key to be searched.

Output Format

The output prints whether the given key is present in the binary search tree or not.

Refer to the sample output for the exact format.

Sample Test Case

Input: 7
10 5 15 3 7 12 20
12
Output: The key 12 is found in the binary search tree

Answer

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

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

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

struct Node* insert(struct Node* 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;
}
```

```
int search(struct Node* root, int key) {
    if (root == NULL)
        return 0;
    if (root->data == key)
        return 1;
    if (key < root->data)
        return search(root->left, key);
    else
        return search(root->right, key);
}

int main() {
    int n, key, val;
    scanf("%d", &n);
    struct Node* root = NULL;
    for (int i = 0; i < n; i++) {
        scanf("%d", &val);
        root = insert(root, val);
    }
    scanf("%d", &key);

    if (search(root, key))
        printf("The key %d is found in the binary search tree\n", key);
    else
        printf("The key %d is not found in the binary search tree\n", key);

    return 0;
}
```

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

2. Problem Statement

Arun is working on a Binary Search Tree (BST) data structure. His goal is to implement a program that reads a series of integers and inserts them into a BST. Once the integers are inserted, he needs to add a given integer value to each node in the tree and find the maximum value in the BST.

Your task is to help Arun implement this program.

Input Format

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

The second line consists of N space-separated integers, each representing an element to be inserted into the BST.

The third line consists of an integer add, representing the value to be added to each node in the BST.

Output Format

The output prints the maximum value in the BST after adding the add value.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 5 15 20 25
5
Output: 30

Answer

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

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

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

struct Node* insert(struct Node* 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;
}

void addToEachNode(struct Node* root, int add) {
    if (root == NULL) return;
    root->data += add;
    addToEachNode(root->left, add);
    addToEachNode(root->right, add);
}

int findMax(struct Node* root) {
    if (root == NULL) return -1;
    while (root->right != NULL)
        root = root->right;
    return root->data;
}

int main() {
    int N, val, add;
    scanf("%d", &N);
    struct Node* root = NULL;
    for (int i = 0; i < N; i++) {
        scanf("%d", &val);
        root = insert(root, val);
    }
    scanf("%d", &add);

    addToEachNode(root, add);

    int maxVal = findMax(root);
    printf("%d\n", maxVal);

    return 0;
}
```

Status : Correct *Marks : 10/10*

3. Problem Statement

Dhruv is working on a project where he needs to implement a Binary Search Tree (BST) data structure and perform various operations on it.

He wants to create a program that allows him to build a BST, traverse it in different orders (inorder, preorder, postorder), and exit the program when needed.

Help Dhruv by designing a program that fulfils his requirements.

Input Format

The first input consists of the choice.

If the choice is 1, enter the number of elements N and the elements inserted into the tree, separated by a space in a new line.

If the choice is 2, print the in-order traversal.

If the choice is 3, print the pre-order traversal.

If the choice is 4, print the post-order traversal.

If the choice is 5, exit.

Output Format

The output prints the results based on the choice.

For choice 1, print "BST with N nodes is ready to use" where N is the number of nodes inserted.

For choice 2, print the in-order traversal of the BST.

For choice 3, print the pre-order traversal of the BST.

For choice 4, print the post-order traversal of the BST.

For choice 5, the program exits.

If the choice is greater than 5, print "Wrong choice".

Refer to the sample output for the formatting specifications.

Sample Test Case

Input: 1
5
12 78 96 34 55
2
3
4
5
Output: BST with 5 nodes is ready to use
BST Traversal in INORDER
12 34 55 78 96
BST Traversal in PREORDER
12 78 34 55 96
BST Traversal in POSTORDER
55 34 96 78 12

Answer

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

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

// Create a new node
struct Node* createNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

// Insert into BST
struct Node* insert(struct Node* 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);
    // Ignore duplicates
    return root;
}

// Inorder traversal
void inorder(struct Node* root) {
    if (root) {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}

// Preorder traversal
void preorder(struct Node* root) {
    if (root) {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}

// Postorder traversal
void postorder(struct Node* root) {
    if (root) {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}

// Free BST
void freeBST(struct Node* root) {
    if (root) {
        freeBST(root->left);
        freeBST(root->right);
        free(root);
    }
}
```

```
int main() {
    struct Node* root = NULL;
    int choice, N, val, i;
    while (1) {
        if (scanf("%d", &choice) != 1) break;
        if (choice == 1) {
            scanf("%d", &N);
            root = NULL; // Clear previous BST if any
            for (i = 0; i < N; i++) {
                scanf("%d", &val);
                root = insert(root, val);
            }
            printf("BST with %d nodes is ready to use\n", N);
        } else if (choice == 2) {
            printf("BST Traversal in INORDER\n");
            inorder(root);
            printf("\n");
        } else if (choice == 3) {
            printf("BST Traversal in PREORDER\n");
            preorder(root);
            printf("\n");
        } else if (choice == 4) {
            printf("BST Traversal in POSTORDER\n");
            postorder(root);
            printf("\n");
        } else if (choice == 5) {
            break;
        } else {
            printf("Wrong choice\n");
        }
    }
    freeBST(root);
    return 0;
}
```

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