Rajalakshmi Engineering College

Name: Harshavarthini
Email: 240701181@rajalakshmi.edu.in
Roll no: 240701181
Phone: 9150394958
Branch: REC
Department: I CSE AH
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

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

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>

// Define the structure for a BST node
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

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

// Insert a value into the BST
struct Node* insert(struct Node* root, int val) {
    if (root == NULL) {
        return createNode(val);
    }
```

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

// Add a value to all nodes in the BST
void addToAllNodes(struct Node* root, int addVal) {
    if (root == NULL) return;
    root->data += addVal;
    addToAllNodes(root->left, addVal);
    addToAllNodes(root->right, addVal);
}

// Find the maximum value in the BST
int findMax(struct Node* root) {
    struct Node* current = root;
    while (current->right != NULL) {
        current = current->right;
    }
    return current->data;
}

// Main function
int main() {
    int N, val, addVal;

    scanf("%d", &N);

    struct Node* root = NULL;

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

    scanf("%d", &addVal);

    addToAllNodes(root, addVal);
```

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

    return 0;
}
```

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

2. 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

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>

// Define the node structure
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

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

// Insert into BST
struct Node* insert(struct Node* root, int val) {
    if (root == NULL) return createNode(val);

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

    return root;
}

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

```
}

// Main function
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***

3. Problem Statement

Jake is learning about binary search trees(BST) and their operations. He wants to implement a program that can delete a node from a BST based on the given key value and print the remaining nodes in an in-order traversal.

Assist Jake in the program.

Input Format

The first line of input consists of an integer n, representing the number of elements in BST.

The second line consists of n space-separated integers, representing the elements of the tree.

The third line consists of an integer x, representing the key value of the node to be deleted.

Output Format

The first line of output prints "Before deletion: " followed by the in-order traversal of the initial BST.

The second line prints "After deletion: " followed by the in-order traversal after the deletion of the key value.

If the key value is not present in the BST, print the original tree as it is.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
8 6 4 3 1
4
Output: Before deletion: 1 3 4 6 8
After deletion: 1 3 6 8

Answer

```
// You are using GCC
#include <stdio.h>
#include <stdlib.h>

// Define the structure for a BST node
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* newNode(int data) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = data;
    temp->left = temp->right = NULL;
```

```
    return temp;
}

// Function to insert a node into BST
struct Node* insert(struct Node* root, int data) {
    if (root == NULL)
        return newNode(data);
    if (data < root->data)
        root->left = insert(root->left, data);
    else if (data > root->data)
        root->right = insert(root->right, data);
    return root;
}

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

// Find the minimum value node in BST
struct Node* minValueNode(struct Node* node) {
    struct Node* current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}

// Delete a node from BST
struct Node* deleteNode(struct Node* 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 {
        // Node with only one child or no child
```

```
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }

        // Node with two children
        struct Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

// Search function to check if key is in BST
int search(struct Node* root, int key) {
    if (root == NULL)
        return 0;
    if (key == root->data)
        return 1;
    else if (key < root->data)
        return search(root->left, key);
    else
        return search(root->right, key);
}

// Main function
int main() {
    int n, x;
    scanf("%d", &n);
    int i, value;
    struct Node* root = NULL;

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

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

    printf("Before deletion: ");
    inorder(root);
    printf("\n");

    if (search(root, x))
        root = deleteNode(root, x);

    printf("After deletion: ");
    inorder(root);
    printf("\n");

    return 0;
}
```

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