Rajalakshmi Engineering College

Name: Prithika S
Email: 240701400@rajalakshmi.edu.in
Roll no: 240701400
Phone: 9790212894
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

Kishore is studying data structures, and he is currently working on implementing a binary search tree (BST) and exploring its basic operations. He wants to practice creating a BST, inserting elements into it, and performing a specific operation, which is deleting the minimum element from the tree.

Write a program to help him perform the delete operation.

Input Format

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

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

Output Format

The output prints the remaining elements of the BST in ascending order (in-order traversal) after deleting the minimum element.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 6
5 3 8 2 4 6
Output: 3 4 5 6 8

Answer

```
// You are using GCC
#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;
}

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;
}

struct Node* deleteMin(struct Node* root) {
    if (root == NULL)
        return NULL;
    if (root->left == NULL) {
        struct Node* rightChild = root->right;
        free(root);
        return rightChild;
    }
    root->left = deleteMin(root->left);
    return root;
}

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

int main() {
    int n, i, val;
    scanf("%d", &n);
    struct Node* root = NULL;
    for (i = 0; i < n; i++) {
        scanf("%d", &val);
        root = insert(root, val);
    }
    root = deleteMin(root);
    inOrder(root);
    printf("\n");
    return 0;
}
```

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

2. Problem Statement

You are given a series of magic levels (integers) and need to construct a Binary Search Tree (BST) from them. After constructing the BST, your task is to perform a range search, which involves finding and printing all the magic levels within a specified range [L, R].

Input Format

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

The second line consists of N space-separated integers, representing the magic levels to insert.

The third line consists of two integers, L and R, which define the range for the search.

Output Format

The output prints all the magic levels within the range [L, R] in ascending order, separated by spaces.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 5 15 3 7
2 20
Output: 3 5 7 10 15

Answer

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

struct Node {
    int data;
    struct Node* left;
    struct Node* 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 rangeSearch(struct Node* root, int L, int R) {
    if (root == NULL)
        return;
    if (L < root->data)
        rangeSearch(root->left, L, R);
    if (L <= root->data && root->data <= R)
        printf("%d ", root->data);
    if (R > root->data)
        rangeSearch(root->right, L, R);
}

int main() {
    int n, i, val, L, R;
    scanf("%d", &n);
    struct Node* root = NULL;
    for (i = 0; i < n; i++) {
        scanf("%d", &val);
        root = insert(root, val);
    }
    scanf("%d %d", &L, &R);
    rangeSearch(root, L, R);
    printf("\n");
    return 0;
}
```

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

3. 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>
```

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

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

int main() {
    int n, i, val, add;
    scanf("%d", &n);
    struct Node* root = NULL;
```

```
    for (i = 0; i < n; i++) {
        scanf("%d", &val);
        root = insert(root, val);
    }
    scanf("%d", &add);
    addValue(root, add);
    int max = findMax(root);
    printf("%d\n", max);
    return 0;
}
```

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