Rajalakshmi Engineering College

Name: Rayvan Sanjai
Email: 240701425@rajalakshmi.edu.in
Roll no: 2116240701425
Phone: 9380572043
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

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

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
   int key;
   struct Node *left, *right;
} Node;

Node* createNode(int key) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   if (!newNode) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->key = key;
   newNode->left = newNode->right = NULL;
   return newNode;
}

Node* insert(Node* root, int key) {
   if (root == NULL) return createNode(key);

   if (key < root->key)
      root->left = insert(root->left, key);
   else if (key > root->key)
      root->right = insert(root->right, key);
```

```
    return root;
}

void rangeSearch(Node* root, int L, int R) {
    if (!root) return;

    if (L < root->key)
        rangeSearch(root->left, L, R);

    if (root->key >= L && root->key <= R)
        printf("%d ", root->key);

    if (R > root->key)
        rangeSearch(root->right, L, R);
}

void freeBST(Node* root) {
    if (!root) return;
    freeBST(root->left);
    freeBST(root->right);
    free(root);
}

int main() {
    int N, value, L, R;
    scanf("%d", &N);

    Node* root = NULL;

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

    scanf("%d %d", &L, &R);

    rangeSearch(root, L, R);
    printf("\n");

    freeBST(root);
    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

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

```
#include <stdlib.h>
typedef struct Node {
   int key;
   struct Node *left, *right;
} Node;

Node* createNode(int key) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   if (!newNode) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->key = key;
   newNode->left = newNode->right = NULL;
   return newNode;
}

Node* insert(Node* root, int key) {
   if (root == NULL) return createNode(key);

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

   return root;
}

int search(Node* root, int key) {
   if (root == NULL) return 0;
   if (root->key == key) return 1;

   return (key < root->key) ? search(root->left, key) : search(root->right, key);
}

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

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

   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);
   freeBST(root);
   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

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
   int key;
   struct Node *left, *right;
} Node;

Node* createNode(int key) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   if (!newNode) {
      printf("Memory allocation failed\n");
      exit(1);
   }
   newNode->key = key;
   newNode->left = newNode->right = NULL;
   return newNode;
}

Node* insert(Node* root, int key) {
   if (root == NULL) return createNode(key);

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

   return root;
}
```

```
void addValue(Node* root, int add) {
    if (root) {
        root->key += add;
        addValue(root->left, add);
        addValue(root->right, add);
    }
}

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

void freeBST(Node* root) {
    if (root) {
        freeBST(root->left);
        freeBST(root->right);
        free(root);
    }
}
int main() {
    int N, value, add;
    scanf("%d", &N);
    Node* root = NULL;
    for (int i = 0; i < N; i++) {
        scanf("%d", &value);
        root = insert(root, value);
    }
    scanf("%d", &add);
    addValue(root, add);
    printf("%d\n", findMax(root));
    freeBST(root);
    return 0;
}
```

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