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 5

Attempt : 1
Total Mark : 10
Marks Obtained : 10

Section 1 : Coding

1. Problem Statement

In his computer science class, John is learning about Binary Search Trees (BST). He wants to build a BST and find the maximum value in the tree.

Help him by writing a program to insert nodes into a BST and find the maximum value in the tree.

Input Format

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

The second line consists of N space-separated integers, representing the values of the nodes to insert into the BST.

Output Format

The output prints the maximum value in the BST.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 5
10 5 15 2 7

Output: 15

Answer

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

struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
struct TreeNode* createNode(int key) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct
TreeNode));
    newNode->data = key;
    newNode->left = newNode->right = NULL;
    return newNode;
}
// You are using GCC
#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 find the maximum value in the BST
int findMaxValue(struct TreeNode* root) {
    if (root == NULL) {
        return INT_MIN; // Return the smallest possible integer if the tree is empty
    }

    // Traverse to the rightmost node to find the maximum value
    while (root->right != NULL) {
        root = root->right;
    }
    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 nodes
    scanf("%d", &n);

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

    // Find the maximum value in the BST
    int maxValue = findMaxValue(root);
    printf("%d\n", maxValue);

    freeTree(root); // Free the allocated memory
    return 0;
}

int main() {
    int N, rootValue;
    scanf("%d", &N);

    struct TreeNode* root = NULL;

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

    int maxVal = findMax(root);
    if (maxVal != -1) {
        printf("%d", maxVal);
    }

    return 0;
}
```

Status : Correct *Marks : 10/10*