Rajalakshmi Engineering College

Name: Siddesh Kumar L
Email: 240701512@rajalakshmi.edu.in
Roll no:
Phone: null
Branch: REC
Department: I CSE FE
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>

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

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

struct Node* insert(struct Node* root, int value) {
   if (root == NULL)
      return createNode(value);
   if (value < root->data)
      root->left = insert(root->left, value);
   else
      root->right = insert(root->right, value);
```

```
    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) {
    while (root->right != NULL)
        root = root->right;
    return root->data;
}

int main() {
    int N, i, value, add;
    struct Node* root = NULL;

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

    scanf("%d", &add);

    addToEachNode(root, add);

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

    return 0;
}
```

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

2. 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

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

```
#include <stdlib.h>

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

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

struct Node* insert(struct Node* root, int value) {
   if (root == NULL)
      return createNode(value);
   if (value < root->data)
      root->left = insert(root->left, value);
   else
      root->right = insert(root->right, value);
   return root;
}

struct Node* findMin(struct Node* root) {
   while (root->left != NULL)
      root = root->left;
   return root;
}

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 {
      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;
        }

        struct Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

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

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

int main() {
    int n, i, x, val;
    struct Node* root = NULL;

    scanf("%d", &n);

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

    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***

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

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

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

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

struct Node* insert(struct Node* root, int value) {
   if (root == NULL)
      return createNode(value);
   if (value < root->data)
      root->left = insert(root->left, value);
   else
      root->right = insert(root->right, value);
   return root;
}

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

void preOrder(struct Node* root) {
   if (root == NULL)
      return;
   printf("%d ", root->data);
```

```
    preOrder(root->left);
    preOrder(root->right);
}

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

int main() {
    int choice, N, i, value;
    struct Node* root = NULL;

    while (1) {
        scanf("%d", &choice);

        switch (choice) {
            case 1:
            root = NULL;
            scanf("%d", &N);
            for (i = 0; i < N; i++) {
                scanf("%d", &value);
                root = insert(root, value);
            }
                printf("BST with %d nodes is ready to use\n", N);
            break;

            case 2:
                printf("BST Traversal in INORDER\n");
                inOrder(root);
                printf("\n");
                break;

            case 3:
                printf("BST Traversal in PREORDER\n");
                preOrder(root);
                printf("\n");
                break;
            case 4:
                printf("BST Traversal in POSTORDER\n");
                postOrder(root);
                printf("\n");
                break;

            case 5:
                exit(0);

            default:
                printf("Wrong choice\n");
        }
    }

    return 0;
}
```

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