Rajalakshmi Engineering College

Name: Shivani R J
Email: 240701500@rajalakshmi.edu.in
Roll no: 2116240701500
Phone: 9962571492
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

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*create(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 create(value);
   if(value<root->data)
   root->left=insert(root->left,value);
```

```
    else if(value>root->data)
    root->right=insert(root->right,value);
    return root;
}

void search(struct node*root,int l,int r)
{
    if(root==NULL)
        return;
    if(l < root->data)
        search(root->left, l, r);
    if(l<=root->data && root->data<=r)
        printf("%d ",root->data);
    if(r > root->data)
        search(root->right,l,r);
}

int main()
{
    int n,i,l,r,value;
    scanf("%d",&n);
    struct node*root=NULL;
    for(i=0;i<n;i++)
    {
        scanf("%d",&value);
        root=insert(root,value);
    }
    scanf("%d %d",&l,&r);
    search(root,l,r);
    printf("\n");
    return 0;
}
```

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

2. 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*create(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 create(value);
   }
   if(value<root->data)
   {
       root->left=insert(root->left,value);
   }
   else if(value>root->data)
   {
       root->right=insert(root->right,value);
   }
   return root;
}

void addele(struct node*root,int add)
{
   if(root==NULL)
   {
       return;
   }
   root->data+=add;
   addele(root->left,add);
   addele(root->right,add);
}

int max(struct node*root)
{
   if(root==NULL)
   {
       return -1;
   }
   while(root->right!=NULL)
   {
       root=root->right;
   }
   return root->data;
}

int main()
{
   struct node*root=NULL;
   int n,i,add;
   scanf("%d",&n);
   int a[n];
   for(i=0;i<n;i++)
   {
       scanf("%d",&a[i]);
       root=insert(root,a[i]);
   }
   scanf("%d",&add);
   addele(root,add);
   printf("%d",max(root));
   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>

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

struct node*create(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 create(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)
   {
      inorder(root->left);
      printf("%d ",root->data);
      inorder(root->right);
   }
}

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

node*del(struct node*root,int key)
{
   if(root==NULL)
   {
      return root;
   }
   if(key<root->data)
   {
      root->left=del(root->left,key);
   }
   else if(key>root->data)
   {
      root->right=del(root->right,key);
   }
   else
   {
      if(root->left==NULL)
      {
          struct node*temp=root->right;
          free(root);
          return temp;
      }
      if(root->right==NULL)
      {
          struct node*temp=root->left;
          free(root);
          return temp;
      }
      struct node*temp=min(root->right);
      root->data=temp->data;
      root->right=del(root->right,temp->data);
   }
   return root;
}

int main()
{
   struct node*root=NULL;
   int n,i,key;
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
      int value;
      scanf("%d",&value);
      root=insert(root,value);
   }
   scanf("%d",&key);
   printf("Before deletion:");
   inorder(root);
   printf("\n");
   root=del(root,key);
   printf("After deletion:");
   inorder(root);
   printf("\n");
   return 0;
}
```

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