Rajalakshmi Engineering College

Name: Nikhilesh D
Email: 240701360@rajalakshmi.edu.in
Roll no: 240701360
Phone: 9940617256
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.

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

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

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

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

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

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

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

int main()
{
    int N, i, L, R;
    scanf("%d", &N);
    int arr[N];
    for (i = 0; i < N; i++)
        scanf("%d", &arr[i]);
    scanf("%d %d", &L, &R);

    Node* root = NULL;
    for (i = 0; i < N; i++)
        root = insert(root, arr[i]);

    printRange(root, L, R);
    return 0;
}
```

Status : Correct   *Marks : 10/10*

2. Problem Statement

Emily is studying binary search trees (BST). She wants to write a program that inserts characters into a BST and then finds and prints the minimum and maximum values.

Guide her with the program.

Input Format

The first line of input consists of an integer N, representing the number of values to be inserted into the BST.

The second line consists of N space-separated characters.

Output Format

The first line of output prints "Minimum value: " followed by the minimum value of the given inputs.

The second line prints "Maximum value: " followed by the maximum value of the given inputs.

Refer to the sample outputs for formatting specifications.

Sample Test Case

Input: 5
Z E W T Y
Output: Minimum value: E
Maximum value: Z

Answer

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

typedef struct Node
{
    char data;
    struct Node* left;
    struct Node* right;
} Node;

Node* createNode(char data)
{
```

```c
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}

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

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

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

int main()
{
    int N;
    scanf("%d", &N);
    Node* root = NULL;
    char ch;

    for (int i = 0; i < N; i++)
    {
        scanf(" %c", &ch); // space before %c to consume any whitespace
        root = insert(root, ch);
```

```
    }

    printf("Minimum value: %c\n", findMin(root));
    printf("Maximum value: %c\n", findMax(root));

    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>

typedef struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
```

```c
} Node;

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

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

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

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

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

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

    while (1)
    {
        scanf("%d", &choice);
        if (choice == 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);
        } else if (choice == 2)
        {
            printf("BST Traversal in INORDER\n");
            inorder(root);
            printf("\n");
        } else if (choice == 3)
        {
            printf("BST Traversal in PREORDER\n");
            preorder(root);
            printf("\n");
        } else if (choice == 4)
        {
            printf("BST Traversal in POSTORDER\n");
            postorder(root);
            printf("\n");
        } else if (choice == 5)
        {
            break;
        } else
        {
            printf("Wrong choice\n");
        }
```

```
    }
    return 0;
}
```

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