Rajalakshmi Engineering College

Name: Soameshwaran D
Email: 240701520@rajalakshmi.edu.in
Roll no: 240701520
Phone: 7358671540
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

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>

// BST Node structure
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

// Insert function
struct Node* insert(struct Node* root, int data) {
    if (root == NULL)
        return newNode(data);
    if (data < root->data)
        root->left = insert(root->left, data);
    else if (data > root->data)
        root->right = insert(root->right, data);
    return root;
}

// In-order traversal
void inorder(struct Node* root) {
    if (root == NULL)
        return;
    inorder(root->left);
    printf("%d ", root->data);
```

```
    inorder(root->right);
}

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

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

int main() {
    struct Node* root = NULL;
    int choice;

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

        if (choice == 1) {
            int N, val;
            scanf("%d", &N);
            root = NULL; // Clear the previous tree
            for (int i = 0; i < N; i++) {
                scanf("%d", &val);
                root = insert(root, val);
            }
            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");
```

240701520

```
        }
        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**

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

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

// Define the Node structure for the BST
struct Node {
    char data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* newNode(char data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

// Insert function to insert a new node into the BST
struct Node* insert(struct Node* root, char data) {
    if (root == NULL)
        return newNode(data);

    // If data is smaller, insert into the left subtree
    if (data < root->data)
```

```
        root->left = insert(root->left, data);
    // If data is greater, insert into the right subtree
    else if (data > root->data)
        root->right = insert(root->right, data);

    return root;
}

// Function to find the minimum value in the BST
char findMin(struct Node* root) {
    while (root != NULL && root->left != NULL) {
        root = root->left;
    }
    return root->data;
}

// Function to find the maximum value in the BST
char findMax(struct Node* root) {
    while (root != NULL && root->right != NULL) {
        root = root->right;
    }
    return root->data;
}

int main() {
    int N;

    // Read the number of nodes to insert into the BST
    scanf("%d", &N);

    char values[N];

    // Read the characters to insert into the BST
    for (int i = 0; i < N; i++) {
        scanf(" %c", &values[i]);  // Read each character with a space before to
ignore any newline
    }

    // Construct the BST
    struct Node* root = NULL;
    for (int i = 0; i < N; i++) {
        root = insert(root, values[i]);
```

240701520

```
    }

    // Find and print the minimum and maximum values in the BST
    printf("Minimum value: %c\n", findMin(root));
    printf("Maximum value: %c\n", findMax(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>

// Define the Node structure for the BST
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* newNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

// Insert function to insert a new node into the BST
struct Node* insert(struct Node* root, int data) {
    if (root == NULL)
        return newNode(data);

    // If data is smaller, insert into the left subtree
    if (data < root->data)
        root->left = insert(root->left, data);
    // If data is greater, insert into the right subtree
    else if (data > root->data)
        root->right = insert(root->right, data);
```

```
    return root;
}

// Function to perform an in-order traversal of the BST
void inorderTraversal(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}

// Function to find the node with the minimum value in the BST
struct Node* findMin(struct Node* root) {
    while (root != NULL && root->left != NULL) {
        root = root->left;
    }
    return root;
}

// Function to delete a node from the BST
struct Node* deleteNode(struct Node* root, int key) {
    // Base case: If the tree is empty
    if (root == NULL)
        return root;

    // Recurse down the tree
    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else {
        // Node to be deleted found

        // Case 1: Node has no children (leaf node)
        if (root->left == NULL && root->right == NULL) {
            free(root);
            return NULL;
        }

        // Case 2: Node has only one child
```

240701520

```
        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;
        }

        // Case 3: Node has two children
        else {
            // Get the in-order successor (smallest in the right subtree)
            struct Node* temp = findMin(root->right);

            // Replace the node's data with the in-order successor's data
            root->data = temp->data;

            // Delete the in-order successor
            root->right = deleteNode(root->right, temp->data);
        }
    }

    return root;
}

int main() {
    int n, x;

    // Read the number of nodes in the BST
    scanf("%d", &n);

    int values[n];

    // Read the elements to insert into the BST
    for (int i = 0; i < n; i++) {
        scanf("%d", &values[i]);
    }

    // Read the key value to be deleted
    scanf("%d", &x);
```

```
    // Construct the BST
    struct Node* root = NULL;
    for (int i = 0; i < n; i++) {
        root = insert(root, values[i]);
    }

    // Before deletion: Print in-order traversal
    printf("Before deletion: ");
    inorderTraversal(root);
    printf("\n");

    // Delete the node with the given key value
    root = deleteNode(root, x);

    // After deletion: Print in-order traversal
    printf("After deletion: ");
    inorderTraversal(root);
    printf("\n");

    return 0;
}
```

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