Rajalakshmi Engineering College

Name: Lakshmi Narayanan S
Email: 241801133@rajalakshmi.edu.in
Roll no: 241801133
Phone: 9345832054
Branch: REC
Department: I AI & DS - AE
Batch: 2028
Degree: B.E - AI & DS

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>

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

// Create new BST node
Node* newNode(int data) {
    Node* node = (Node*) malloc(sizeof(Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

// Insert data in BST
Node* insert(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;
}

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

// Preorder traversal
```

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

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

// Free BST nodes
void freeBST(Node* root) {
    if (root == NULL) return;
    freeBST(root->left);
    freeBST(root->right);
    free(root);
}

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

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

        if (choice == 1) {
            // Build BST
            int n;
            scanf("%d", &n);
            freeBST(root);
            root = NULL;

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

```
            printf("BST with %d nodes is ready to use\n", n);
        }
        else if (choice == 2) {
            if (root == NULL) {
                // No nodes inserted yet
                printf("BST Traversal in INORDER\n");
            } else {
                printf("BST Traversal in INORDER\n");
                inorder(root);
                printf("\n");
            }
        }
        else if (choice == 3) {
            if (root == NULL) {
                printf("BST Traversal in PREORDER\n");
            } else {
                printf("BST Traversal in PREORDER\n");
                preorder(root);
                printf("\n");
            }
        }
        else if (choice == 4) {
            if (root == NULL) {
                printf("BST Traversal in POSTORDER\n");
            } else {
                printf("BST Traversal in POSTORDER\n");
                postorder(root);
                printf("\n");
            }
        }
        else if (choice == 5) {
            // Exit program
            freeBST(root);
            break;
        }
        else {
            // Wrong choice
            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>
```

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

// Create new node
Node* newNode(char data) {
    Node* node = (Node*) malloc(sizeof(Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}

// Insert a character into BST
Node* insert(Node* root, char 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;
}

// Find minimum value in BST (leftmost node)
char findMin(Node* root) {
    Node* current = root;
    while (current->left != NULL)
        current = current->left;
    return current->data;
}

// Find maximum value in BST (rightmost node)
char findMax(Node* root) {
    Node* current = root;
    while (current->right != NULL)
        current = current->right;
    return current->data;
}

// Free the BST nodes
void freeBST(Node* root) {
```

```
    if (root == NULL) return;
    freeBST(root->left);
    freeBST(root->right);
    free(root);
}

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

    Node* root = NULL;
    for (int i = 0; i < n; i++) {
        char val;
        scanf(" %c", &val);  // space before %c to consume any whitespace
        root = insert(root, val);
    }

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

    freeBST(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>

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

// Create a new node
Node* newNode(int data) {
    Node* node = (Node*) malloc(sizeof(Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
```

```
}

// Insert a node into BST
Node* insert(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;
}

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

// Find minimum node in the BST (used in deletion)
Node* minValueNode(Node* root) {
    Node* current = root;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}

// Delete a node with given key from BST
Node* deleteNode(Node* root, int key, int *found) {
    if (root == NULL) return root;

    if (key < root->data) {
        root->left = deleteNode(root->left, key, found);
    } else if (key > root->data) {
        root->right = deleteNode(root->right, key, found);
    } else {
        // Node found
        *found = 1;

        // Case 1: No child or one child
        if (root->left == NULL) {
```

```
        Node* temp = root->right;
        free(root);
        return temp;
    }
    else if (root->right == NULL) {
        Node* temp = root->left;
        free(root);
        return temp;
    }

    // Case 2: Two children
    Node* temp = minValueNode(root->right);
    root->data = temp->data;
    root->right = deleteNode(root->right, temp->data, found);
  }
  return root;
}

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

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

  int key;
  scanf("%d", &key);

  printf("Before deletion: ");
  inorder(root);
  printf("\n");

  int found = 0;
  root = deleteNode(root, key, &found);

  printf("After deletion: ");
  inorder(root);
  printf("\n");
```

```
    return 0;
}
```

Status : Correct *Marks : 10/10*