Rajalakshmi Engineering College

Name: naveen prasath
Email: 240701352@rajalakshmi.edu.in
Roll no: 240701352
Phone: 9585322006
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 : 23.5

Section 1 : Coding

1. Problem Statement

Kishore is studying data structures, and he is currently working on implementing a binary search tree (BST) and exploring its basic operations. He wants to practice creating a BST, inserting elements into it, and performing a specific operation, which is deleting the minimum element from the tree.

Write a program to help him perform the delete operation.

Input Format

The first line of input consists of an integer N, representing the number of elements Kishore wants to insert into the BST.

The second line consists of N space-separated integers, where each integer represents an element to be inserted into the BST.

Output Format

The output prints the remaining elements of the BST in ascending order (in-order traversal) after deleting the minimum element.

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 6
5 3 8 2 4 6
Output: 3 4 5 6 8

Answer

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

```
Node* findMin(Node* root) {
    while (root && root->left != NULL) {
        root = root->left;
    }
    return root;
}
Node* deleteMin(Node* root) {
    if (root == NULL) {
        return NULL;
    }
    if (root->left == NULL) {
        Node* rightChild = root->right;
        free(root);
        return rightChild;
    }
    root->left = deleteMin(root->left);
    return root;
}
void inorderTraversal(Node* root) {
    if (root == NULL) return;
    inorderTraversal(root->left);
    printf("%d ", root->data);
    inorderTraversal(root->right);
}

int main() {
    int N;
    scanf("%d", &N);
    int elements[N];
    for (int i = 0; i < N; i++) {
        scanf("%d", &elements[i]);
    }
    Node* root = NULL;
    for (int i = 0; i < N; i++) {
        root = insert(root, elements[i]);
    }
    root = deleteMin(root);
    inorderTraversal(root);
    printf("\n");

    return 0;
}
```

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

2. 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

```
#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);
    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);
}
void freeTree(Node* root) {
    if (root == NULL) return;
    freeTree(root->left);
    freeTree(root->right);
    free(root);
}

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

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

        if (choice == 1) {
            int N;
            scanf("%d", &N);
            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) {
        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) {
        freeTree(root);
        break;
    } else {
        printf("Wrong choice\n");
    }
  }

  return 0;
}
```

Status : Partially correct ***Marks : 3.5/10***

3. 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

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    char data;
    struct Node* left;
    struct Node* right;
} Node;
Node* createNode(char data) {
    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 && root->left != NULL) {
        root = root->left;
    }
    return root->data;
}
char findMax(Node* root) {
    while (root && root->right != NULL) {
        root = root->right;
    }
    return root->data;
}

int main() {
    int N;
    scanf("%d", &N);
    char arr[N];

    for (int i = 0; i < N; i++) {
        scanf(" %c", &arr[i]);
    }
    Node* root = NULL;
    for (int i = 0; i < N; i++) {
        root = insert(root, arr[i]);
    }
    printf("Minimum value: %c\n", findMin(root));
    printf("Maximum value: %c\n", findMax(root));

    return 0;
}
```

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