Rajalakshmi Engineering College

Name: NEIL DANIEL A
Email: 240701356@rajalakshmi.edu.in
Roll no: 240701356
Phone: 8925059757
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

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>

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

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

struct Node* insert(struct 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;
}

struct Node* deleteMin(struct Node* root) {
    if (root == NULL) {
        return NULL;
    }
    if (root->left == NULL) {
        struct Node* rightChild = root->right;
        free(root);
        return rightChild;
    }
    root->left = deleteMin(root->left);
    return root;
}

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

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

    int data;
    struct Node* root = NULL;

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

    root = deleteMin(root);

    inorderTraversal(root);
    printf("\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

```
#include <stdio.h>
```

```
#include <stdlib.h>

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

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

struct Node* insert(struct 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(struct Node* root) {
    while (root && root->left != NULL) {
        root = root->left;
    }
    return root->data;
}

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

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

    char data;
    struct Node* root = NULL;

    for (int i = 0; i < N; i++) {
        scanf(" %c", &data);
        root = insert(root, data);
    }

    char minVal = findMin(root);
    char maxVal = findMax(root);

    printf("Minimum value: %c\n", minVal);
    printf("Maximum value: %c\n", maxVal);

    return 0;
}
```

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

3. Problem Statement

Edward has a Binary Search Tree (BST) and needs to find the k-th largest element in it.

Given the root of the BST and an integer k, help Edward determine the k-th largest element in the tree. If k exceeds the number of nodes in the BST, return an appropriate message.

Input Format

The first line of input consists of integer n, the number of nodes in the BST.

The second line consists of the n elements, separated by space.

The third line consists of the value of k.

Output Format

The output prints the kth largest element in the binary search tree.

For invalid inputs, print "Invalid value of k".

Refer to the sample output for formatting specifications.

Sample Test Case

Input: 7
8 4 12 2 6 10 14
1

Output: 14

Answer

```
#include <stdio.h>
#include <stdlib.h>

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

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

struct Node* insert(struct 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 findKthLargestUtil(struct Node* root, int* k, int* result) {
    if (root == NULL || *k <= 0) {
        return;
    }

    findKthLargestUtil(root->right, k, result);

    if (*k == 1) {
        *result = root->data;
    }

    (*k)--;

    findKthLargestUtil(root->left, k, result);
}

int findKthLargest(struct Node* root, int k) {
    int result = -1;
    findKthLargestUtil(root, &k, &result);
    return result;
}

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

    int data;
    struct Node* root = NULL;

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

    scanf("%d", &k);

    int kthLargest = findKthLargest(root, k);

    if (kthLargest == -1) {
        printf("Invalid value of k\n");
    } else {
        printf("%d\n", kthLargest);
    }

    return 0;
}
```

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