

# Rajalakshmi Engineering College

Name: Vithyavarshini P  
Email: 240801387@rajalakshmi.edu.in  
Roll no: 240801387  
Phone: 9361308913  
Branch: REC  
Department: I ECE AF  
Batch: 2028  
Degree: B.E - ECE

Scan to verify results



## NeoColab\_REC\_CS23231\_DATA STRUCTURES

### REC\_DS using C\_Week 5\_COD\_Question 1

Attempt : 1  
Total Mark : 10  
Marks Obtained : 10

#### Section 1 : Coding

##### 1. Problem Statement

John is learning about Binary Search Trees (BST) in his computer science class. He wants to create a program that allows users to delete a node with a given value from a BST and print the remaining nodes using an in-order traversal.

Implement a function to help him delete a node with a given value from a BST.

##### ***Input Format***

The first line of input consists of an integer N, representing the number of nodes in the BST.

The second line consists of N space-separated integers, representing the values of the BST nodes.

The third line consists of an integer V, which is the value to delete from the BST.

### **Output Format**

The output prints the space-separated values in the BST in an in-order traversal, after the deletion of the specified value.

If the specified value is not available in the tree, print the given input values in-order traversal.

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: 5  
10 5 15 2 7  
15  
Output: 2 5 7 10

### **Answer**

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

```
struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
```

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

```
struct TreeNode* insert(struct TreeNode* root, int data)
{
    if (root == NULL)
```

```
{
    return createNode(data);
}
if (data < root->data)
{

    root->left = insert(root->left, data);

} else if (data > root->data)
{

    root->right = insert(root->right, data);

}
return root;
}
```

```
struct TreeNode* findMin(struct TreeNode* node)
{

    struct TreeNode* current = node;
    while (current && current->left != NULL)
    {

        current = current->left;

    }
    return current;
}
```

```
}
```

```
struct TreeNode* deleteNode(struct TreeNode* root, int key)
```

```
{
```

```
    if (root == NULL)
```

```
{
```

```
        return root;
```

```
}
```

```
    if (key < root->data)
```

```
{
```

```
        root->left = deleteNode(root->left, key);
```

```
    } else if (key > root->data)
```

```
{
```

```
        root->right = deleteNode(root->right, key);
```

```
    } else
```

```
{
```

```
        // Node with the key found
```

```
        // Case 1: Node with only one child or no child
```

```
    if (root->left == NULL)
    {

        struct TreeNode* temp = root->right;
        free(root);
        return temp;

    } else if (root->right == NULL)

    {

        struct TreeNode* temp = root->left;
        free(root);
        return temp;

    }

    // Case 2: Node with two children
    struct TreeNode* minRight = findMin(root->right);
    root->data = minRight->data;
    root->right = deleteNode(root->right, minRight->data);
}
return root;

}
```

```
void inorderTraversal(struct TreeNode* root)
```

```
{

    if (root != NULL)
```

```
{
```

```
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);

    }

}

int main()
{
    int N, rootValue, V;
    scanf("%d", &N);
    struct TreeNode* root = NULL;
    for (int i = 0; i < N; i++) {
        int key;
        scanf("%d", &key);
        if (i == 0) rootValue = key;
        root = insert(root, key);
    }
    scanf("%d", &V);
    root = deleteNode(root, V);
    inorderTraversal(root);
    return 0;
}
```

**Status :** Correct

**Marks :** 10/10