LAB SUBMISSION – 7 <-link

Meher Shrishti Nigam – 20BRS1193

1- Write a program to insert the following element in a binary tree
12,45,23,8,11,9
Once a binary tree forms then search an element 10 in the tree and return appropriate message.

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

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

typedef struct BinaryTree{
    Node * root;
}BinaryTree;

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

/*
 *  Creates a binary tree structure with one node. Root pointer points to that node.
*/

BinaryTree * createBinaryTree(int item)
{
    BinaryTree * bt = (BinaryTree *)malloc(sizeof(BinaryTree));
    Node * newnode = createNode(item);
    bt->root = newnode;
    return bt;
```

```
}

/*
 * Inserts node at left of the node specified unless node is already
present.
*/

void insertAtLeft(Node * root, int item)
{
    Node * newnode = createNode(item);
    if(root->left == NULL)
        root->left = newnode;
    else
        printf("Left node already occupied.\n");
}

/*
 * Inserts node at right of the node specified unless node is already
present.
*/

void insertAtRight(Node * root, int item)
{
    Node * newnode = createNode(item);
    if(root->right == NULL)
        root->right = newnode;
    else
        printf("Right node already occupied.\n");
}

int is_found = 0;
void search(int item, Node * root)
{
    if(root == NULL)
        return;
    if(root->data == item)
    {
        printf("Item %d was found.\n", item);
        is_found = 1;
        return;
    }
    search(item, root->left);
    search(item, root->right);
```

```
}

int main()
{
    BinaryTree * bt = createBinaryTree(12);
    insertAtLeft(bt->root, 45);
    insertAtLeft(bt->root->left, 23);
    insertAtRight(bt->root, 8);
    insertAtRight(bt->root->left, 11);
    insertAtRight(bt->root->right, 9);
    is_found = 0;
    search(10, bt->root);
    if(is_found == 0)
        printf("Item was not found.\n");
}
```

```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_7> gcc BinaryTree.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_7> ./a.exe
Item was not found.
```

2- Write a menu driven program to
   a. Insert the n element in the tree following the binary tree fashion of insertion.
   b. Traverse the elements of tree using
      i. In-order traversal technique
      ii. Post-order traversal technique
      iii. Pre-order traversal technique
   c. Search a given element (by user) in binary tree
   d. Perform the deletion operation on
      i. Entire tree
      ii. One element from the tree

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

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

```c
typedef struct BinaryTree{
    Node * root;
}BinaryTree;

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

/*
 *  Creates a binary tree structure with one node. Root pointer points
to that node.
*/

BinaryTree * createBinaryTree(int item)
{
    BinaryTree * bt = (BinaryTree *)malloc(sizeof(BinaryTree));
    Node * newnode = createNode(item);
    bt->root = newnode;
    return bt;
}

/*
 * Inserts node at left of the node specified unless node is already
present.
*/

void insertAtLeft(Node * root, int item)
{
    Node * newnode = createNode(item);
    if(root->left == NULL)
        root->left = newnode;
    else
        printf("Left node already occupied.\n");
}

/*
 * Inserts node at right of the node specified unless node is already
present.
```

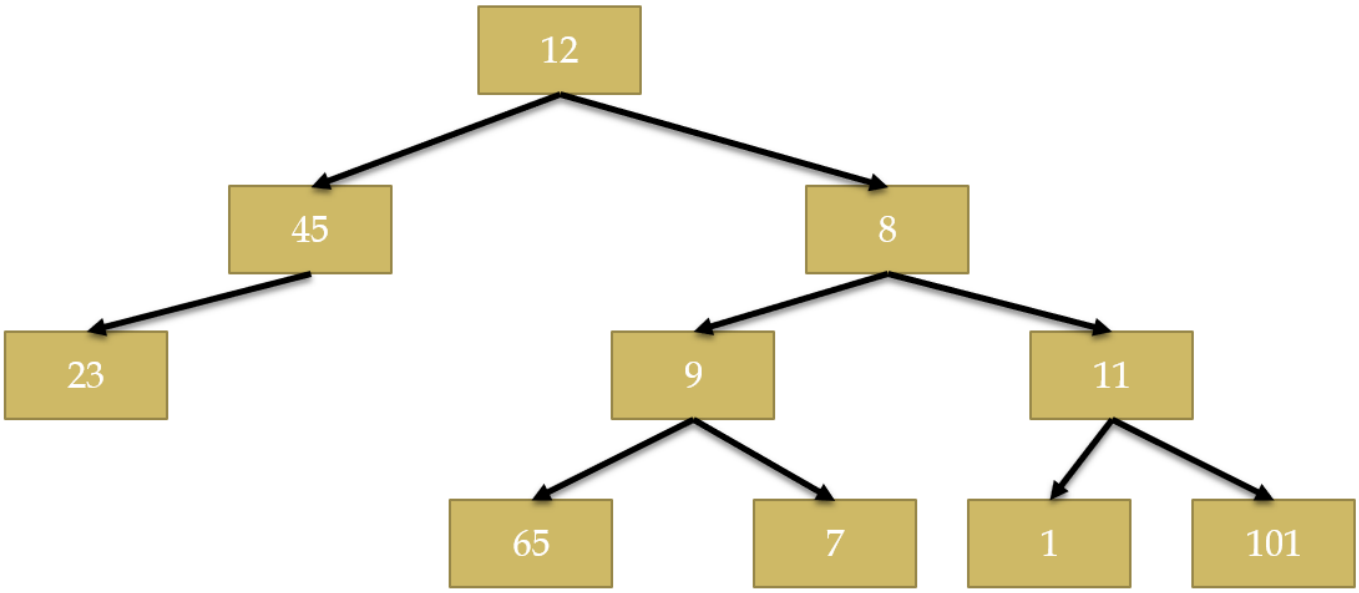
```c
*/

void insertAtRight(Node * root, int item)
{
    Node * newnode = createNode(item);
    if(root->right == NULL)
        root->right = newnode;
    else
        printf("Right node already occupied.\n");
}

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

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

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

void deleteBinaryTree(Node * root)
{
```

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

// Deletes elements using the node of the element to be delted and its
parent node.
void deleteNode(Node * root, Node * parent)
{
    // If 0 child
    if(root->left == NULL && root->right == NULL)
    {
        if(root == parent->right)
            parent->right = NULL;
        else if(root == parent->left)
            parent->left = NULL;
        free(root);
        printf("The node had 0 children and now has been deleted.\n");
        return;
    }
    // If 1 child
    if(root->left != NULL && root->right == NULL)
    {
        if(root == parent->right)
            parent->right = root->left;
        else if(root == parent->left)
            parent->left = root->left;
        printf("The node had 1 child and now has been deleted.\n");
        free(root);
        return;
    }
    if(root->left == NULL && root->right != NULL)
    {
        if(root == parent->right)
            parent->right = root->right;
        else if(root == parent->left)
            parent->left = root->right;
        free(root);
        return;
```

```
    }
    // If 2 child - we delete the rightmost deepest node from root after exchanging the data in the rightmost deepest node from root and the node to be deleted
    Node * temp = root;
    Node * temp2 = parent;
    while(temp->right != NULL)
    {
        temp = temp->right;
        temp2 = temp2->right;
    }
    root->data = temp->data;
    temp2->right = NULL;
    printf("The node had 2 children and now has been deleted.\n");
    free(temp);
}

int is_found = 0;
void search(int item, Node * root)
{
    if(root == NULL)
        return;
    if(root->data == item)
    {
        printf("Item %d was found.\n", item);
        is_found = 1;
        return;
    }
    search(item, root->left);
    search(item, root->right);
}

int main()
{
    BinaryTree * bt = createBinaryTree(12);
    insertAtLeft(bt->root, 45);
    insertAtLeft(bt->root->left, 23);
    insertAtRight(bt->root, 8);
    insertAtRight(bt->root->right, 11);
    insertAtLeft(bt->root->right, 9);
    insertAtRight(bt->root->right->right, 101);
    insertAtLeft(bt->root->right->right, 1);
    insertAtRight(bt->root->right->left, 7);
```

```
    insertAtLeft(bt->root->right->left, 65);

    // Searching in the binary tree
    is_found = 0;
    search(10, bt->root);
    if(is_found == 0)
        printf("Item was not found.\n");
        is_found = 0;
    search(1, bt->root);
    if(is_found == 0)
        printf("Item was not found.\n");

    // Inorder, Preorder and Postorder traversals of the binary tree
    InorderTraversal(bt->root);
    printf("\n");
    PreorderTraversal(bt->root);
    printf("\n");
    PostorderTraversal(bt->root);
    printf("\n");

    // 0 child deletion
    deleteNode(bt->root->right->right->right, bt->root->right->right);
    InorderTraversal(bt->root);
    printf("\n");
    // 1 child deletion
    deleteNode(bt->root->right->right, bt->root->right);
    InorderTraversal(bt->root);
    printf("\n");
    // 2 child deletion
    deleteNode(bt->root->right, bt->root);
    InorderTraversal(bt->root);
    printf("\n");

    // deletion of whole binary tree
    deleteBinaryTree(bt->root);
    InorderTraversal(bt->root);
    printf("\n");
}
```

The binary tree formed looks like –




```
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_7> gcc BinaryTree.c
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_7> ./a.exe
Item was not found.
Item 1 was found.
23 45 12 65 9 7 8 1 11 101
12 45 23 8 9 65 7 11 1 101
23 45 65 7 9 1 101 11 8 12
The node had 0 children and now has been deleted.
23 45 12 65 9 7 8 1 11
The node had 1 child and now has been deleted.
23 45 12 65 9 7 8 1
The node had 2 children and now has been deleted.
23 45 12 65 9 7 1
PS C:\Users\meher\OneDrive\Documents\ASem3\DSA\Lab\Lab_7>
```