Page :
Date : / /

Experiment 3

Aim :- Write a C Program to implement a binary search tree using operations like insertion, deletion, counting of nodes, counting leaf nodes etc.

Objectives :- After study of this experiment, the student will be able to
- To use basic principles of programming as applied to complex data structures.
- To implement the tree through programming.

Outcomes :- After study of this experiment, the student will be able to.
- Formulate and solve problems of binary search trees and it's operations
- Understand the concepts and apply the methods in basic trees.

Prerequisite :- Types of trees, Binary trees.

Requirements :- PC and Turbo C compiler version 3.0

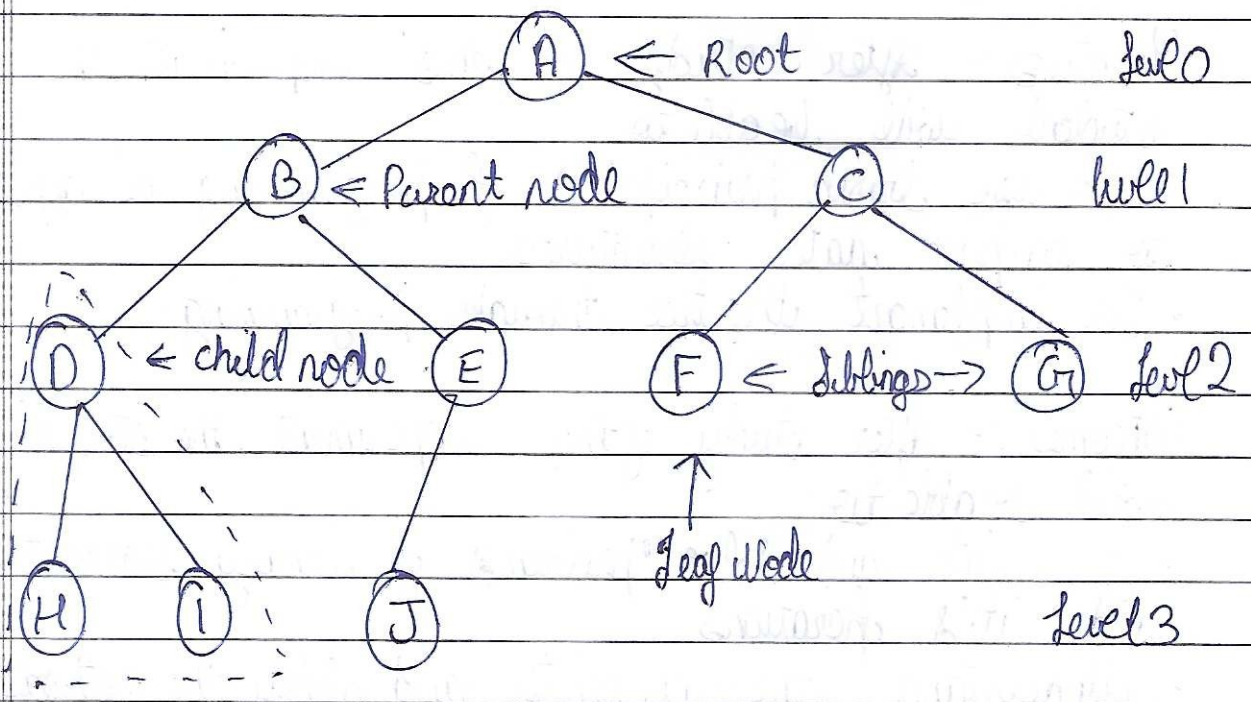
Pre-Experiment Exercise :-

Brief Theory :-

Tree :-
- Tree represents the nodes connected by edges.
- A binary tree is a data structure that is defined

Yash Mahajan SE ITB 04 191061

as a collection of elements called nodes. In a binary tree, the ~~top~~ maximum number of children nodes a node can have is two. A binary tree has the benefits of both an ordered array and a linked list as search is as quick as a sorted array and insertion and deletion operation is as quick as a linked list.



Following are important terms with respect to a tree.

Path :- Path refers to the sequence of nodes along the edges of a tree.

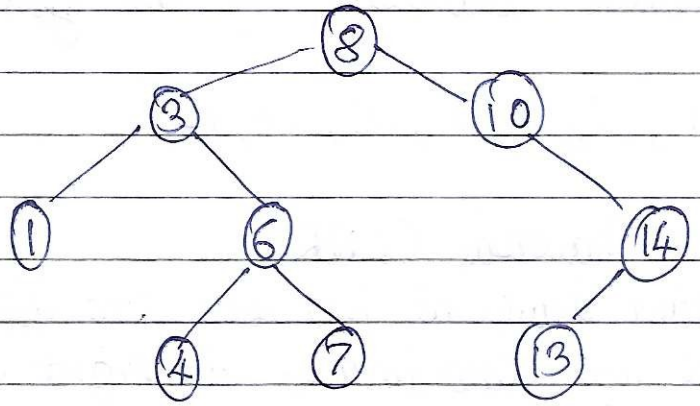
Root - The node at top of the tree is called root. There is only one root per tree and one path from the root node to any node.

Parent :- Any node except the root node ~~to any node~~ has one edge upward to a node called parent.

Child :- The node below a given node connected

its edge downward is called child node.
Leaf:- The node which does not have any child node is called a leaf node.
Sub-tree - Sub-tree represents descendants of a node.
Visiting - Visiting refers to checking the value of a node when control is on the node.
Traversing - Traversing means passing through nodes in a specific order.
Levels - Level of a node represents the generation of node. If root node is at level 0, then it's next child node is at level 1, it's next child at level 2 and so on.
Key's - Key represents a value of a node based on which search operation is to be carried out for a node.

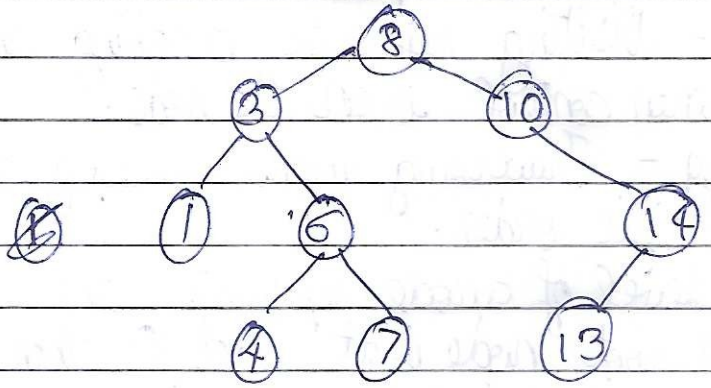
Binary Search Tree.
Every element has a unique key. The keys in a non empty left sub tree (right sub tree) are smaller (larger) than the key of the root of the sub tree. The left and right sub trees are also binary search trees.



Page:
Date: / /

Operations on a binary search Tree :-

- Traversing and displaying a binary search tree.



In Binary search trees we can traverse a tree and display it's elements in 3 ways.

1) Preorder Traversal (NLR)

In the preorder traversal, the root node is processed first, followed by the left sub-tree and then the right sub tree. We first process the root node, then left sub tree and then right sub tree. The left sub tree is in turn processed recursively as is the right sub tree. The word pre in preorder specifies that the root node is accessed first. The preorder traversal of the given tree is

8, 3, 1, 6, 4, 7, 10, 14, 13

2) Inorder Traversal (LNR)

The inorder traversal processes the left sub tree first, then the root, and finally the right sub tree. The prefix in in means that the root is processed in-

between the sub trees. The implementation of the algorithm is done recursively

The inorder traversal of the given tree is :-

1, 3, 4, 6, 7, 8, 10, 13, 14

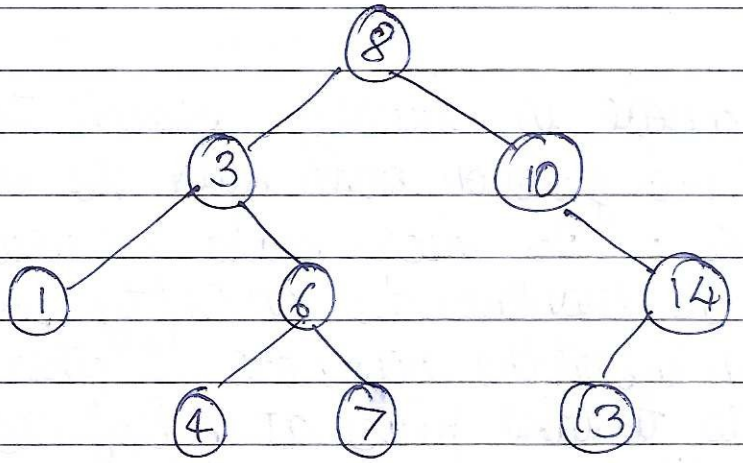
3) Post order traversal (LRN)

The postorder traversal processes the ~~the~~ root node after (post) the left ~~sub~~ and right sub trees have been processed. It starts by locating the far-left leaf node and processing it. It then processes its right sibling including its sub-trees (if any). Finally it processes the root node.

The ~~to~~ postorder ~~to~~ traversal of the given tree is

1, 4, 7, 6, 3, 13, 14, 10, 8

- Searching a node in binary search tree :-

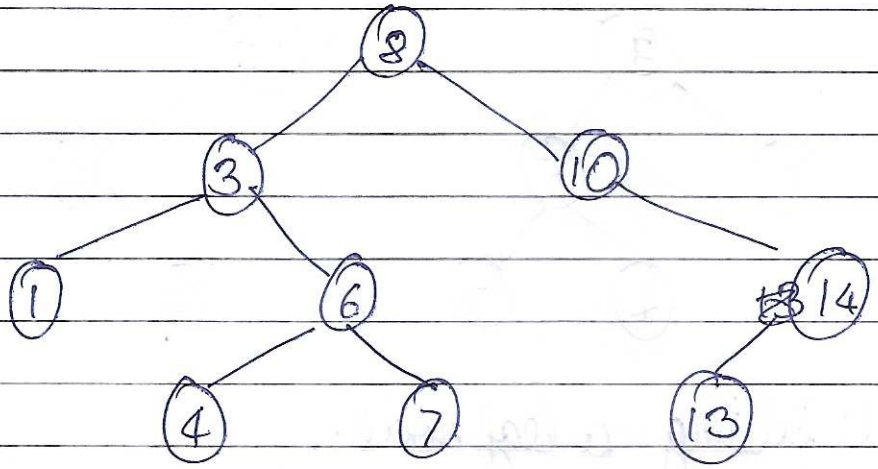


The search function is used to find weather a given value is present in the tree or not. The searching process begins at the root node. If root node is null, that means the binary search tree is empty, and the algorithm terminates. If there are nodes in the tree, then the search function checks if the value to be searched is equal to the key value of current ~~node~~ node. If not, it checks if the value ~~is of the~~ to be searched is less than the key value of first node, in ~~the~~ which case it should be recursively called on the left child node. In case the value is greater ~~than~~ than the value of current node, it should call the search function on right child node recursively.

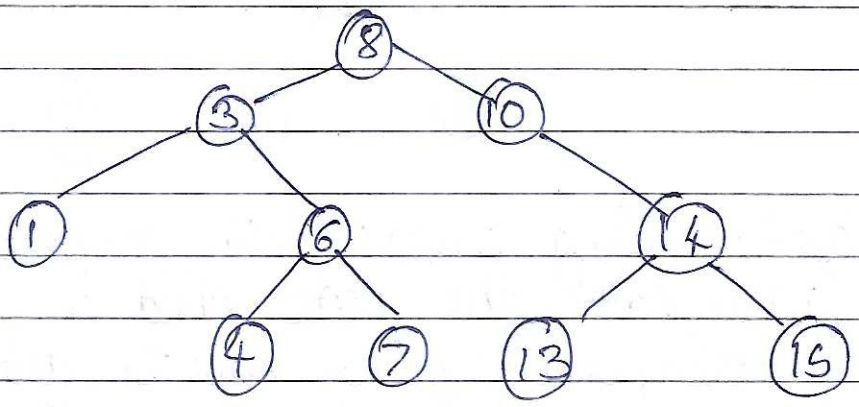
Eg To find value 6, the function first compares it to root node. Since 8 is greater than 6, it will call the function on ~~right~~ left child node. Now 3 is less than 6 so it will call search on right child node of 3, now 6 is equal to 6 so the function will show the required output.

- Insert a node in binary search tree:-
The insert node function adds data to a binary search tree. To insert data we need to follow the branches to an empty sub-tree and then insert the new node. In other words all inserts ~~to~~ take place at a leaf node or at a leaf like node - a node that has only one

null subTree.



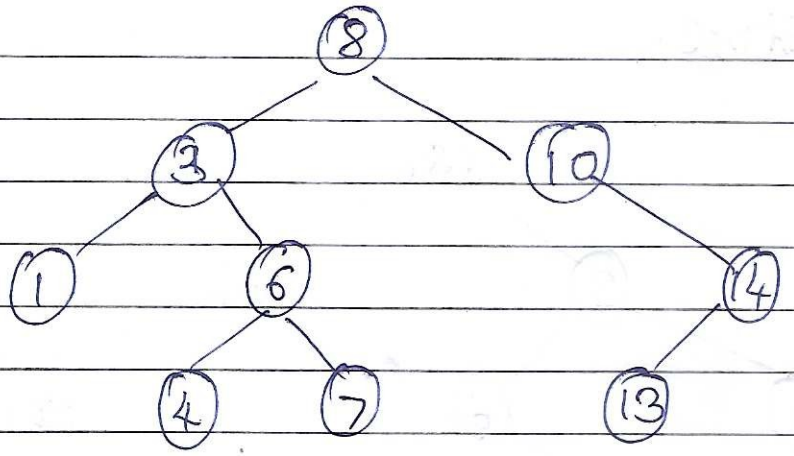
To insert 15 in the given tree, we first compare it with the root node, since 15 is greater than 8, the function is called on the right sub-tree recursively. Now 10 is less than 15, so again insert is called on the right sub tree recursively. Now 14 is less than 15 and the right child node of 14 is null. So 15 is inserted at that position.



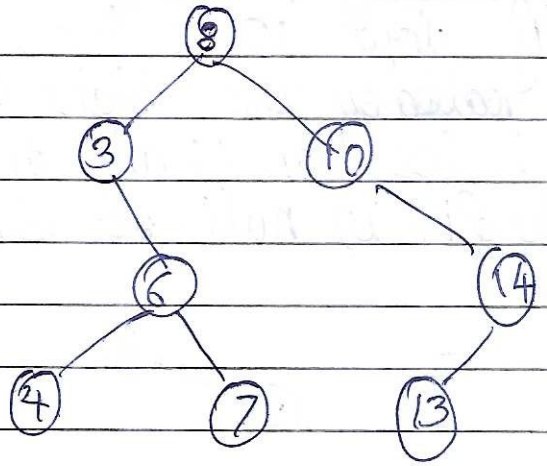
- Deleting a node in binary search tree.
The delete function deletes a node from the binary search tree. However, while deleting nodes, the rules of BST should not be violated.

Yash Mahajan SE IT B 04 191061

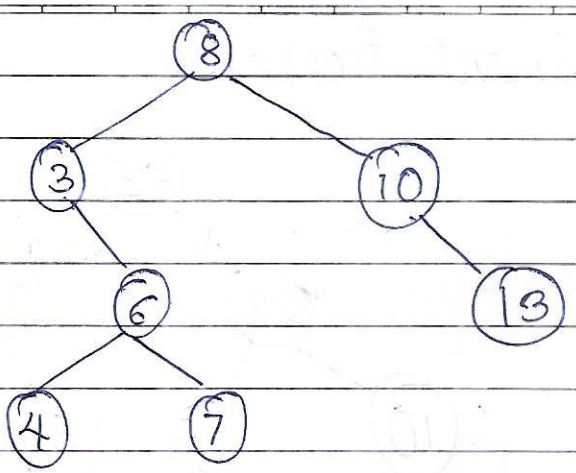
Page:
Date: / /



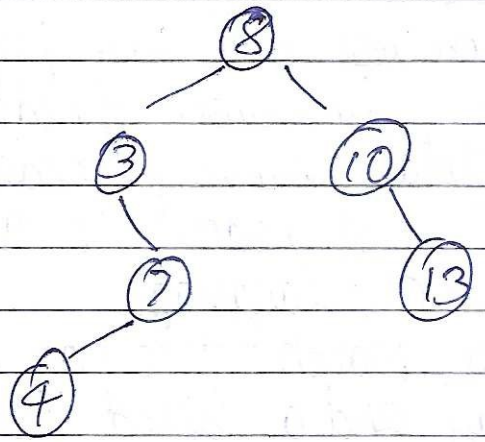
Case 1: Deleting a leaf node.
If we have to remove 1 from the given tree, we can simply remove this node without any issue. This is the simplest case of deletion.



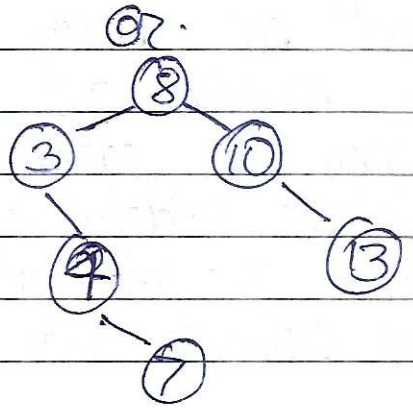
Case 2: Deleting a node with one child.
In this case the node's child is set as child of node's parent. If we have to delete 14 from the tree, we have to make 13 the right child of 10.



Case 3: Deleting a node with 2 children.
In this case we replace the deleted node with it's inorder sucessor or it's inorder predecessor, and then delete the predcessor or sucessor
In the given Tree, to delete 6 we can either replace 6 by 4 or 7 as 4 is it's inorder and 7 is it's inorder sucessor and then delete 4 or 7.



or.



7. Laboratory Exercise

A. Procedure

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

struct node{

    int data;
    struct node * left;
    struct node * right;
};

struct node * tree;
void create_tree(struct node * tree);
struct node * insert(struct node * tree, int value);
struct node * delete(struct node * tree, int value);
void inOrderTraversal(struct node * tree);
int totalNodes(struct node * tree);
int leafNodes(struct node * tree);


int main(){

    int num,value;

    create_tree(tree);
    do
    {
        printf("\nEnter 1 to insert in tree");
        printf("\nEnter 2 to delete a node in tree");
        printf("\nEnter 3 to display (inorder) in tree");
        printf("\nEnter 4 to find total nodes in tree");
        printf("\nEnter 5 to find total leaf nodes in tree");
        printf("\nEnter 6 to exit");
        printf("\nEnter your choice:");
        scanf("%d",&num);
        switch (num)
        {
```

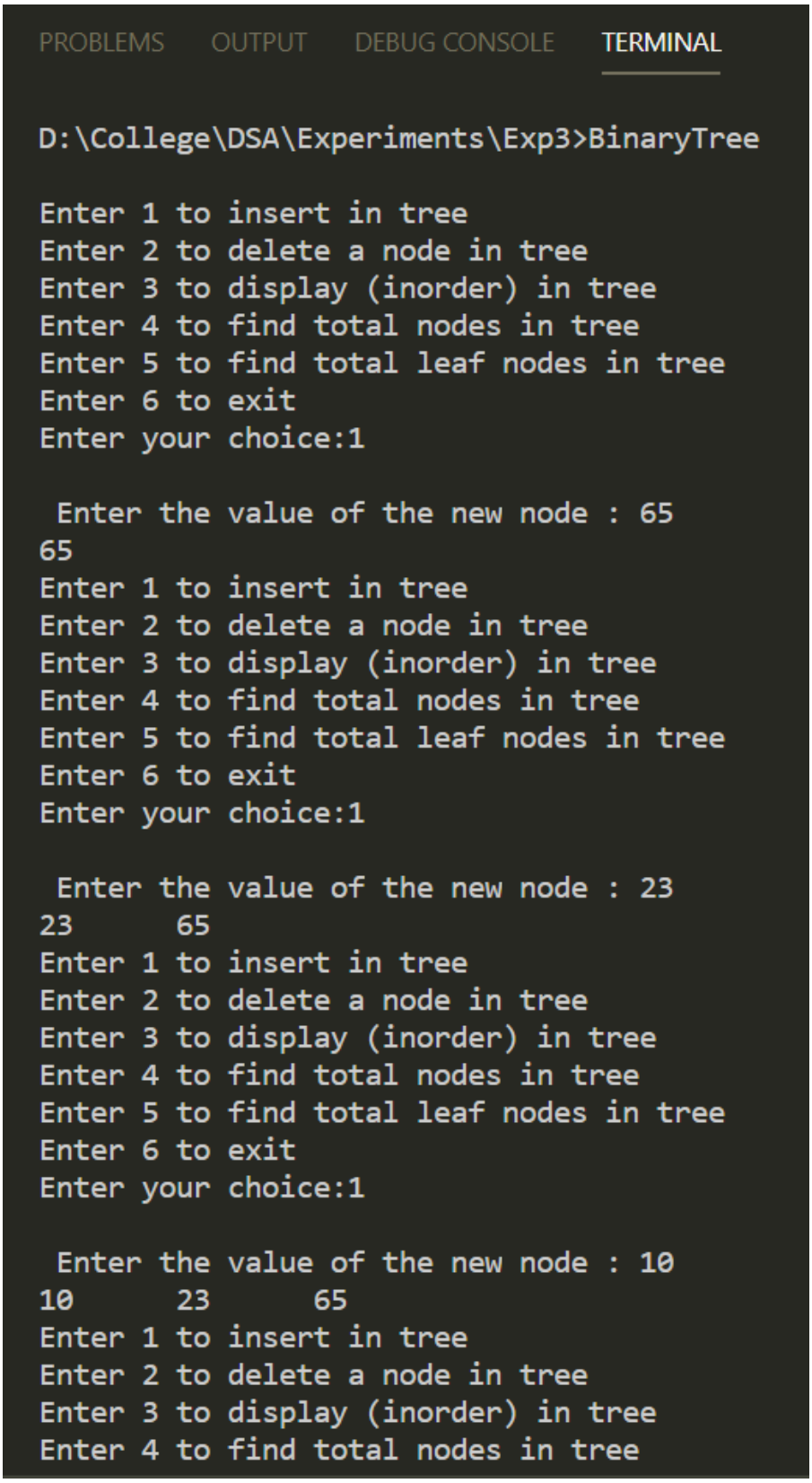
```
        case 1:
            printf("\n Enter the value of the new node : ");
            scanf("%d", &value);
            tree = insert(tree, value);
            break;
        case 2:
            printf("\n Enter the element to be deleted : ");
            scanf("%d", &value);
            tree = delete(tree, value);
            break;
        case 3:
            printf("\n The elements of the tree are : \n");
            inOrderTraversal(tree);
            break;
        case 4:
            printf("\n Total no. of nodes = %d", totalNodes(tree));
            break;
        case 5:
            printf("\n Total no. of external nodes = %d",leafNodes(tree));
            break;

        default:
            break;
        }
    } while (num<6);


    return 0;
}

void create_tree(struct node *tree)
{
    tree = NULL;
}

struct node * insert(struct node * tree, int value){

    struct node *ptr, *nodeptr, *parentptr;
    ptr = (struct node*)malloc(sizeof(struct node));
    ptr->data = value;
```

```
    ptr->left = NULL;
    ptr->right = NULL;

    if (tree == NULL)
    {
        tree = ptr;
    }
    else
    {
        parentptr=NULL;
        nodeptr=tree;
        while(nodeptr!=NULL)
        {
            parentptr=nodeptr;
            if(value< nodeptr->data)
                nodeptr=nodeptr->left;
            else
                nodeptr = nodeptr->right;
        }
        if(value<parentptr->data)
            parentptr->left = ptr;
        else
        parentptr->right = ptr;
    }
    inOrderTraversal(tree);
    return tree;

}

void inOrderTraversal(struct node *tree)
{
    if(tree != NULL)
    {
        inOrderTraversal(tree->left);
        printf("%d\t", tree->data);
        inOrderTraversal(tree->right);
    }

}
```

```
int totalNodes(struct node * tree){
    if(tree == NULL){
        return 0;
    }
    else
    {
        return totalNodes(tree->right)+totalNodes(tree->left)+1;
    }

}
int leafNodes(struct node * tree){

    if(tree==NULL){
        return 0;
    }
    else if((tree->left==NULL) && (tree->right==NULL)){
    return 1;
    }
    else
    return (leafNodes(tree->left) + leafNodes(tree->right));
}

struct node *delete(struct node *tree, int val)
{
    struct node * ptr, *current, *temp;

    //To check if tree is null or not
    if (tree==NULL)
    {
        printf("\nThe Tree is empty.");
        return tree;
    }
    //To check if the value to be deleted exists in the tree
    current = tree;
    while(current!=NULL && val!= current->data)
    {
        current = (val<current->data)? current->left:current->right;
    }
    if(current == NULL)
    {
```

```
        printf("\n The value to be deleted is not present in the tree");
        return(tree);
    }
    //Recursive call to delete function
    //If val<data then delete left subtree
    if (val < tree->data)
    {
        tree->left = delete(tree->left, val);
    }
    //If val>data then delete left subtree
    else if (val > tree->data)
    {
        tree->right = delete(tree->right, val);
    }
    else
    {
        // node with only one child or no child
        if (tree->left == NULL)
        {
            temp = tree->right;
            free(tree);
            return temp;
        }
        else if (tree->right == NULL)
        {
            temp = tree->left;
            free(tree);
            return temp;
        }
        //To find inorder Sucessor
        ptr = tree->right;
        while (ptr && ptr->left != NULL)
            ptr = ptr->left;
        temp = ptr;
        tree->data = temp->data;
        tree->right = delete(tree->right, temp->data);
    }
    return tree;
}
```

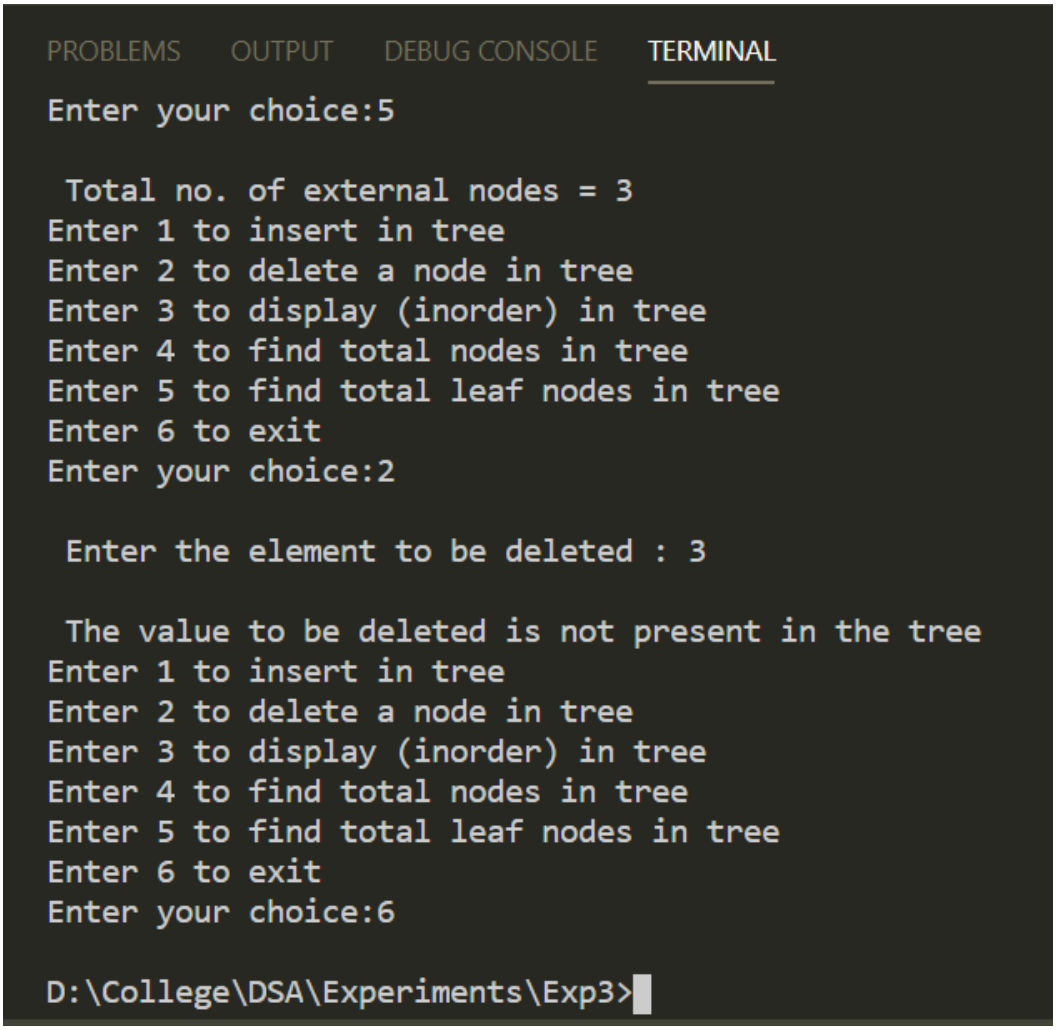
B. Result/Observation/Program code:



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Enter 4 to find total nodes in tree
Enter 5 to find total leaf nodes in tree
Enter 6 to exit
Enter your choice:1

 Enter the value of the new node : 33
10      23      33      65
Enter 1 to insert in tree
Enter 2 to delete a node in tree
Enter 3 to display (inorder) in tree
Enter 4 to find total nodes in tree
Enter 5 to find total leaf nodes in tree
Enter 6 to exit
Enter your choice:1

 Enter the value of the new node : 98
10      23      33      65      98
Enter 1 to insert in tree
Enter 2 to delete a node in tree
Enter 3 to display (inorder) in tree
Enter 4 to find total nodes in tree
Enter 5 to find total leaf nodes in tree
Enter 6 to exit
Enter your choice:2

 Enter the element to be deleted : 33

Enter 1 to insert in tree
Enter 2 to delete a node in tree
Enter 3 to display (inorder) in tree
Enter 4 to find total nodes in tree
Enter 5 to find total leaf nodes in tree
Enter 6 to exit
Enter your choice:1

 Enter the value of the new node : 45
10      23      45      65      98
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
10      23      45      65      98
Enter 1 to insert in tree
Enter 2 to delete a node in tree
Enter 3 to display (inorder) in tree
Enter 4 to find total nodes in tree
Enter 5 to find total leaf nodes in tree
Enter 6 to exit
Enter your choice:1

 Enter the value of the new node : 62
10      23      45      62      65      98
Enter 1 to insert in tree
Enter 2 to delete a node in tree
Enter 3 to display (inorder) in tree
Enter 4 to find total nodes in tree
Enter 5 to find total leaf nodes in tree
Enter 6 to exit
Enter your choice:1

 Enter the value of the new node : 49
10      23      45      49      62      65      98
Enter 1 to insert in tree
Enter 2 to delete a node in tree
Enter 3 to display (inorder) in tree
Enter 4 to find total nodes in tree
Enter 5 to find total leaf nodes in tree
Enter 6 to exit
Enter your choice:4

 Total no. of nodes = 7
Enter 1 to insert in tree
Enter 2 to delete a node in tree
Enter 3 to display (inorder) in tree
Enter 4 to find total nodes in tree
Enter 5 to find total leaf nodes in tree
Enter 6 to exit
Enter your choice:5
```

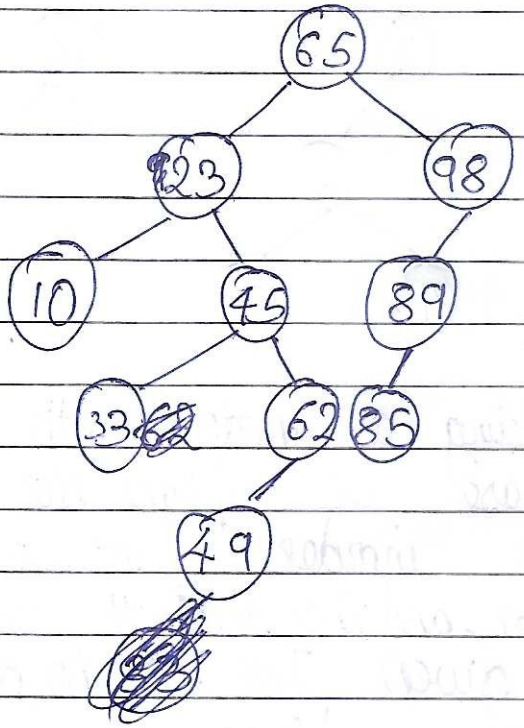

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Enter your choice:5
Total no. of external nodes = 3
Enter 1 to insert in tree
Enter 2 to delete a node in tree
Enter 3 to display (inorder) in tree
Enter 4 to find total nodes in tree
Enter 5 to find total leaf nodes in tree
Enter 6 to exit
Enter your choice:2
Enter the element to be deleted : 3
The value to be deleted is not present in the tree
Enter 1 to insert in tree
Enter 2 to delete a node in tree
Enter 3 to display (inorder) in tree
Enter 4 to find total nodes in tree
Enter 5 to find total leaf nodes in tree
Enter 6 to exit
Enter your choice:6
D:\College\DSA\Experiments\Exp3>

Yash Mahajan SE IT B 04 191061

Page: Date: / /

Post - Experiment Exercise.

A. Question.



Conclusion

In this experiment we have sucessfully implemented a binary search tree and have performed operations like traversal, insertion of node, deletion of node, counting the total number of nodes and counting the number of leaf nodes in a given binary tree using 'C' language.

Binary search tree has benefits of both a sorted array and a linked list. As search is as quick as an array and insertion and deletion is as quick as a linked list. Due to efficiency in search methods, binary search trees are widely used in dictionary problems where the code always inserts and searches the elements that are indexed by some key value.

References:-

1. S.K. Srivastava, Deepali Srivastava ; Data Structures through C in depth ; BPB Publications ; 2011.

2. Reema Thareja ; Data Structures using C ; Oxford.

3. Data Structures ~~using~~ A Pseudocode Approach with C, Richard F. Gilberg & Behrouz A. Forouzan, Second edition, CENGAGE Learning.