K. J. Somaiya College of Engineering, Mumbai-77
(A constituent College of Somaiya Vidyavihar University)

Batch: A3	**Roll No.: 16010121045**

Experiment / assignment / tutorial No. 7

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of BST & Binary tree traversal techniques.

Objective: To Understand and Implement Binary Search Tree, Preorder, Postorder and Inorder Traversal Techniques.

Expected Outcome of Experiment:

CO	Outcome
1	Explain the different data structures used in problem solving

Books/ Journals/ Websites referred:

1. *Fundamentals Of Data Structures In C* – Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications* – Jean Paul Tremblay, Paul G. Sorenson
3. *Data Structures A Pseudo Approach with C* – Richard F. Gilberg & Behrouz A. Forouzan
4. https://www.geeksforgeeks.org/binary-tree-data-structure/
5. https://www.thecrazyprogrammer.com/2015/03/c-program-for-binary-search-tree-insertion.html

Abstract:

A tree is a non- linear data structure used to represent hierarchical relationship existing among several data items. It is a finite set of one or more data items such that, there is a special data item called the root of the tree. Its remaining data items are partitioned into number of mutually exclusive subsets, each of which is itself a tree, and they are called subtrees.

A binary tree is a finite set of nodes. It is either empty or It consists a node called root with two disjoint binary trees-Left subtree, Right subtree. The Maximum degree of any node is 2

A Binary Search Tree is a node-based binary tree data structure in which the left subtree of a node contains only nodes with keys lesser than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key. The left and right subtree each must also be a binary search tree.

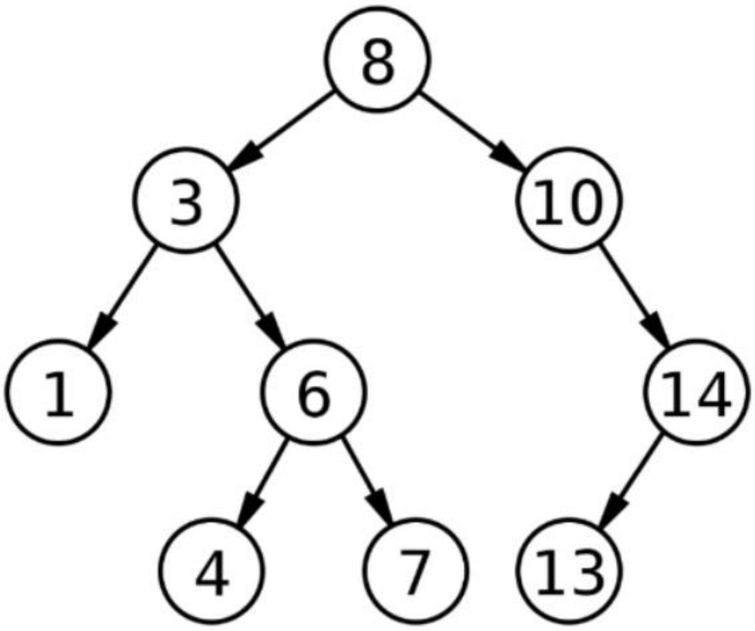
Related Theory: -
Preorder Traversal of BST
Until all nodes are traversed −
Step 1 − Visit root node.
Step 2 − Recursively traverse left subtree. Step 3 − Recursively traverse right subtree.

Postorder Traversal of BST

Until all nodes are traversed −
Step 1 − Recursively traverse left subtree. Step 2 − Recursively traverse right subtree.
Step 3 − Visit root node.

Inorder Traversal of BST
Until all nodes are traversed −
Step 1 − Recursively traverse left subtree. Step 2 − Visit root node.
Step 3 − Recursively traverse right subtree.

Diagram for :



Preorder Traversal of BST

8 3 1 6 4 7 10 14 13

Postorder Traversal of BST

1 4 7 6 3 13 14 10 8

Inorder Traversal of BST

1 3 4 6 7 8 10 13 14

Algorithm for Implementation of BST & Binary tree traversal techniques:

Search in BST: -
1. Start from the root.

2. Compare the searching element with root, if less than root, then recurse for left, else recurse for right.

3. If the element to search is found anywhere, return true, else return false.

Insertion in BST: -

1. Start from the root.

2. Compare the inserting element with root, if less than root, then recurse for left, else recurse for right.

3. After reaching the end, just insert that node at left (if less than current) else right.

Deletion in BST: -

1. Search the node that to be deleted in BST.

2. If node has left or right subtree then find the inorder predecessor or inorder successor respectively (call it new node) and replace the node data value with its data. Then make recursive call to the delete function for that new node.

3 Else delete the node and assign the parent's child pointer(which is node) to NULL.

Implementation Details:

Enlist all the Steps followed and various options explored.

A structure called Node is created which has two struct pointers (left and right) and 1 int variable (data).

A class called BST is created with functions for insertion, deletion and searching. For insertion in BST: -
• If root is null create new node and assign the data to it.
• Else traverse in tree according to the data of node.

For searching: -

• If node is null then the node with required data isn't there in the BST.

• Else traverse in tree according to the data of node.

For deletion: -

• Search the node to be deleted.

• If the node has left or right child then replace its data value with inorder successor (if right child is present) or predecessor of current node. Then search of that node in the corresponding subtree.

• Else delete the node and assign its parents pointer (which points to current node) as NULL.

An object of BST class is created and a menu driven program is made for the user to choose an operation.

Assumptions made for Input:

The value of all nodes are integers and |value|<=INTMAX

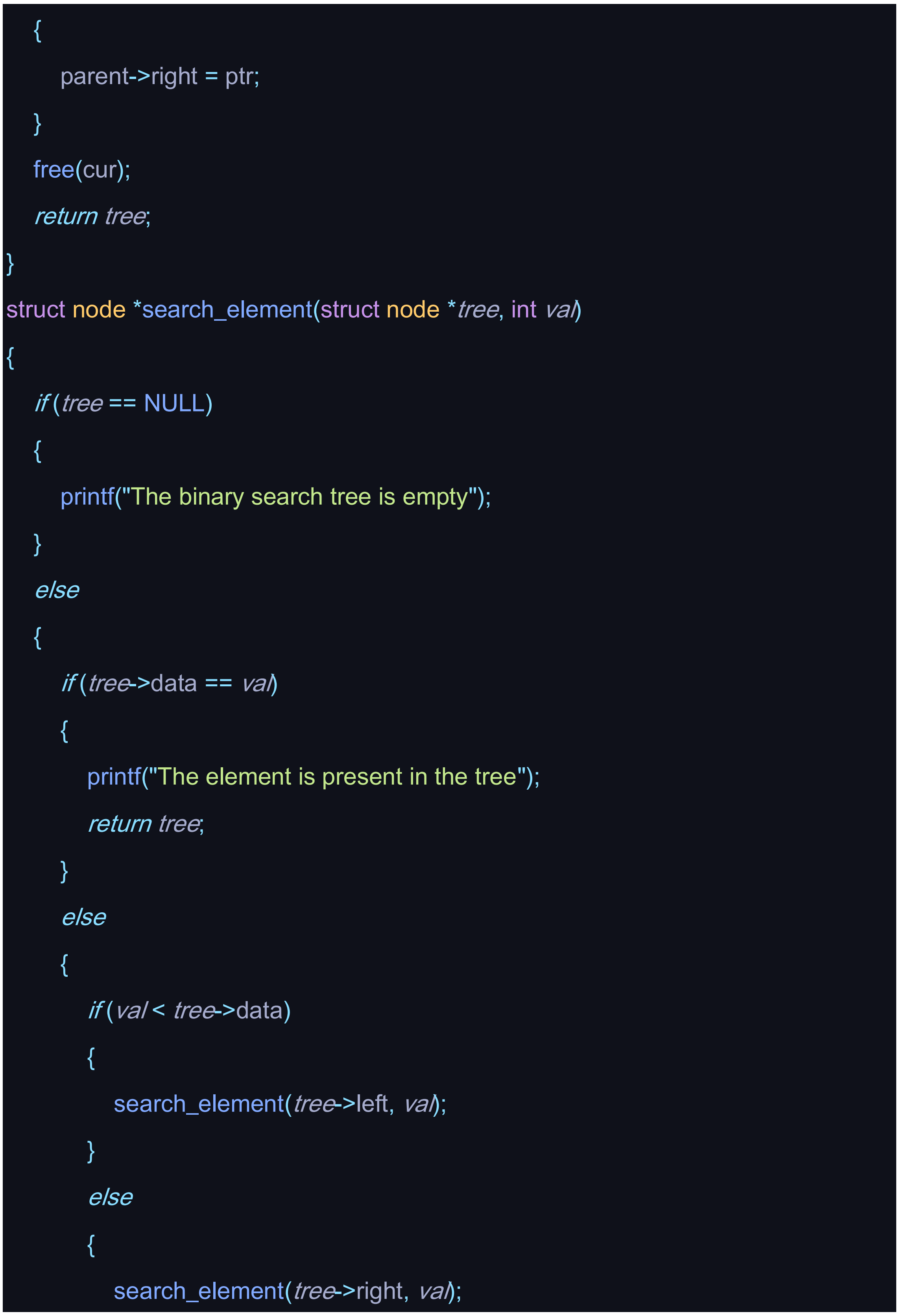
Program source code for Implementation of BST & Binary tree traversal techniques :

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *left;
    struct node *right;
} * tree;
void create_tree(struct node *tree)
{
    tree = NULL;
}
struct node *insert_element(struct node *tree, int val)
{
    struct node *ptr, *nodeptr, *parentptr;
    ptr = (struct node *)malloc(sizeof(struct node));
    ptr->data = val;
    ptr->left = NULL;
    ptr->right = NULL;
    if (tree == NULL)
    {
        tree = ptr;
        tree->left = NULL;
        tree->right = NULL;
    }
    else if (val == tree->data)
```

```
  {
      printf("The value already exists in the tree you cannot insert the same value
again enter a different value of the element\n");
  }
  else
  {
      parentptr = NULL;
      nodeptr = tree;
      while (nodeptr != NULL)
      {
          parentptr = nodeptr;
          if (val < nodeptr->data)
          {
              nodeptr = nodeptr->left;
          }
          else
          {
              nodeptr = nodeptr->right;
          }
      }
      if (val < parentptr->data)
      {
          parentptr->left = ptr;
      }
      else
      {
          parentptr->right = ptr;
      }
```

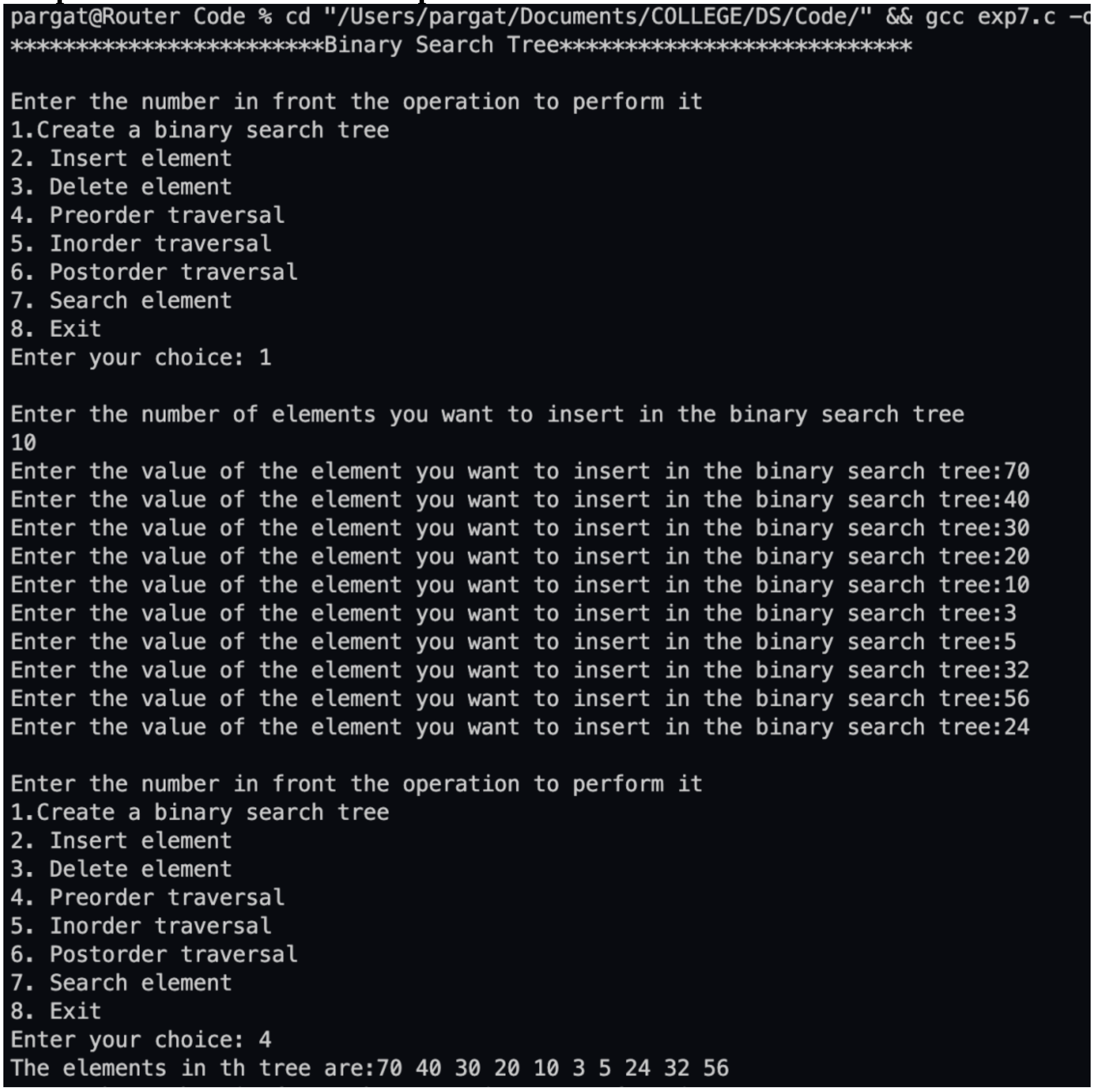
```
    }
    return tree;
}
void preorderTraversal(struct node *tree)
{
    if (tree != NULL)
    {
        printf("%d ", tree->data);
        preorderTraversal(tree->left);
        preorderTraversal(tree->right);
    }
}
void inordeTraversal(struct node *tree)
{
    if (tree != NULL)
    {
        inordeTraversal(tree->left);
        printf("%d\t", tree->data);
        inordeTraversal(tree->right);
    }
}
void postorderTraversal(struct node *tree)
{
    if (tree != NULL)
    {
        postorderTraversal(tree->left);
        postorderTraversal(tree->right);
        printf("%d\t", tree->data);
    }
}
struct node *del_element(struct node *tree, int val)
{
    struct node *ptr, *parent, *cur, *suc, *psuc;
    if (tree == NULL)
    {
        printf("\nThe tree is empty");
        return (tree);
    }
    parent = tree;
    cur = tree->left;
    while (cur != NULL && val != cur->data)
    {
        parent = cur;
        cur = (val < cur->data) ? cur->left : cur->right;
    }
    if (cur == NULL)
    {
        printf("\n The value to be deleted is not present in the tree");
        return (tree);
    }
    if (cur->left == NULL)
    {
        ptr = cur->right;
    }
    else if (cur->right == NULL)
    {
```

```
    ptr = cur->left;
}
else
{
    psuc = cur;
    cur = cur->right;
    while (suc->left != NULL)
    {
        psuc = suc;
        suc = suc->left;
    }
    if (cur == psuc)
    {
        suc->left = cur->left;
    }
    else
    {
        suc->left = cur->left;
        psuc->left = suc->right;
        suc->right = cur->right;
    }
    ptr = suc;
}
if (parent->left == cur)
{
    parent->left = ptr;
}
else
```

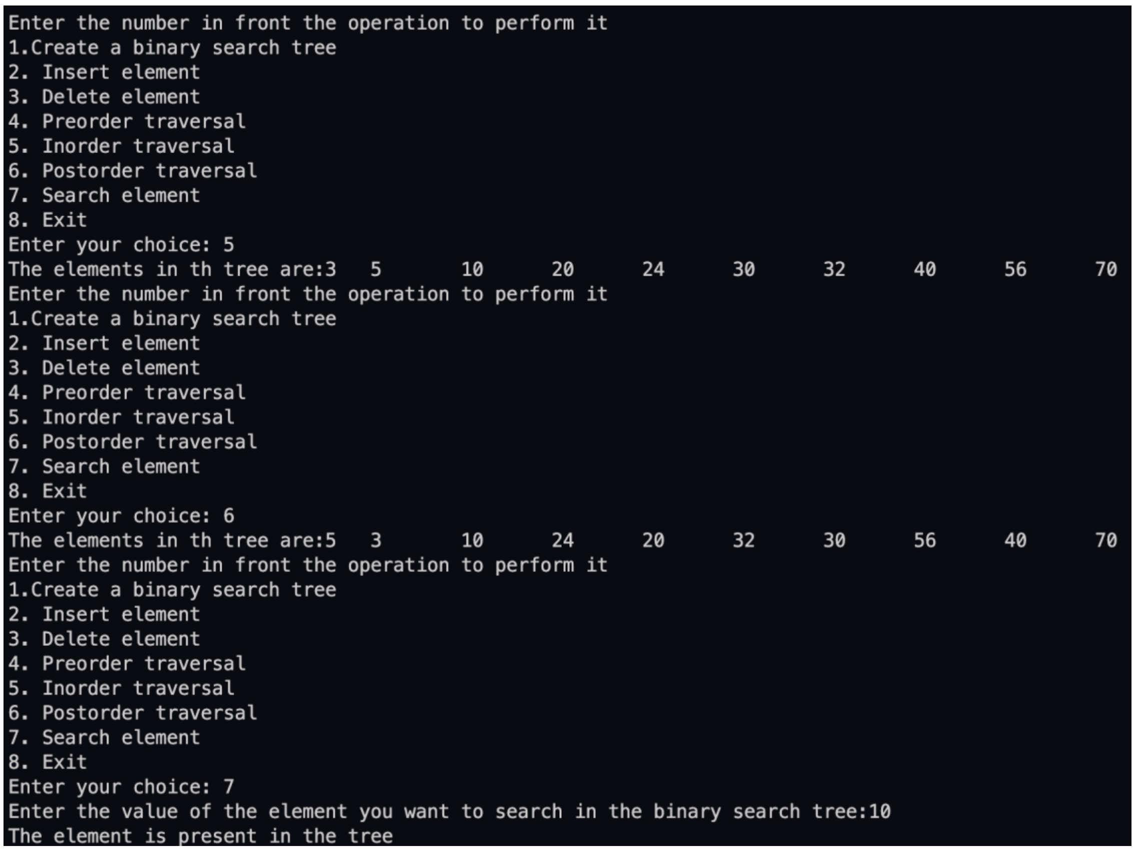


```
    {
        parent->right = ptr;
    }
    free(cur);
    return tree;
}
struct node *search_element(struct node *tree, int val)
{
    if (tree == NULL)
    {
        printf("The binary search tree is empty");
    }
    else
    {
        if (tree->data == val)
        {
            printf("The element is present in the tree");
            return tree;
        }
        else
        {
            if (val < tree->data)
            {
                search_element(tree->left, val);
            }
            else
            {
                search_element(tree->right, val);
            }
        }
    }
}
int main()
{
    int choice, val, n;
    create_tree(tree); printf("************************Binary Search 
Tree***************************\n");
while(1){
        printf("\nEnter the number in front the operation to perform it\n");
        printf("1.Create a binary search tree\n");
        printf("2. Insert element\n");
        printf("3. Delete element\n");
        printf("4. Preorder traversal\n");
        printf("5. Inorder traversal\n");
        printf("6. Postorder traversal\n");
        printf("7. Search element\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("\nEnter the number of elements you want to insert in the binary 
search tree\n");
            scanf("%d", &n);
            for (int i = 1; i <= n; i++)
            {
                printf("Enter the value of the element you want to insert in the binary
search tree:");
                scanf("%d", &val);
                tree = insert_element(tree, val);
            }
            break;
        case 2:
            printf("Enter the value of the element you want to insert in the binary search
tree:");
            scanf("%d", &val);
            tree = insert_element(tree, val);
            break;
        case 3:
            printf("Enter the value of the element you want to delete from the binary
search tree:");
            scanf("%d", &val);
            tree = del_element(tree, val);
            break;
        case 4:
            printf("The elements in th tree are:");
            preorderTraversal(tree);
            break;
        case 5:
            printf("The elements in th tree are:");
            inordeTraversal(tree);
            break;
        case 6:
            printf("The elements in th tree are:");
            postorderTraversal(tree);
            break;
        case 7:
            printf("Enter the value of the element you want to search in the binary
search tree:");
            scanf("%d", &val);
            tree = search_element(tree, val);
            break;
        case 8:
            printf("\nExiting the program");
            exit(1);
            break;
        default:
            printf("\nInvalid choice");
            exit(1);
            break;
        } }
}
```

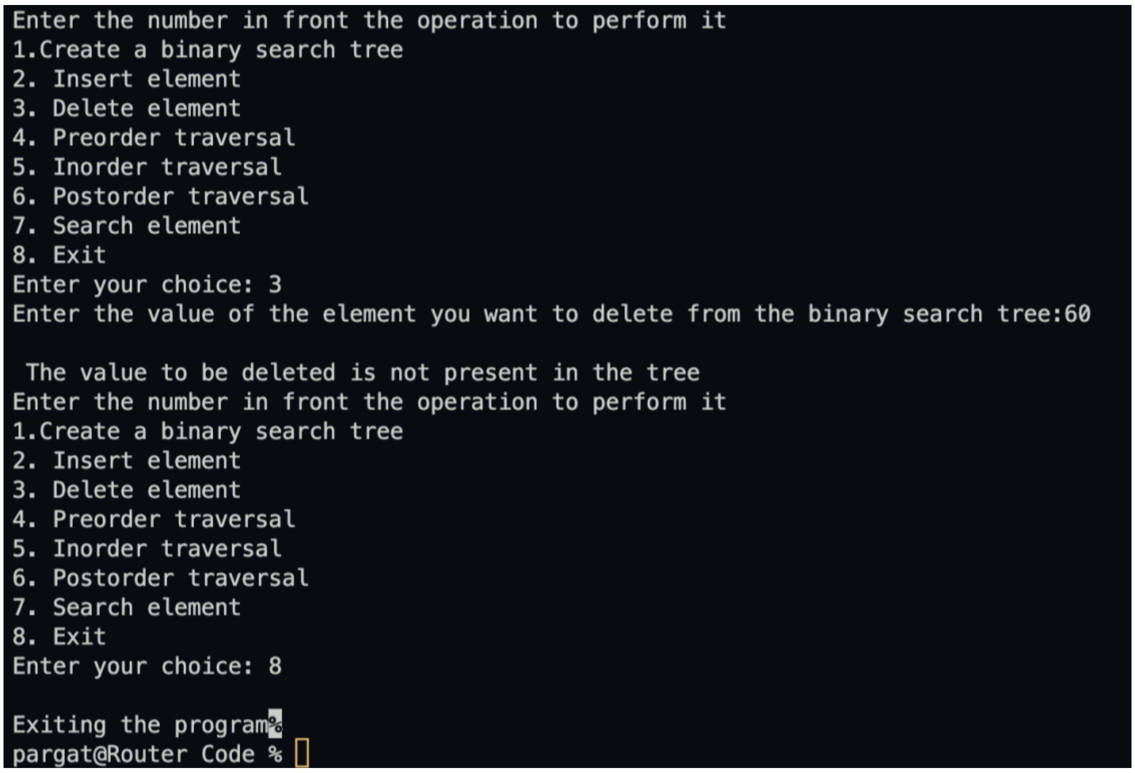
Output Screenshots for Each Operation:



```
pargat@Router Code % cd "/Users/pargat/Documents/COLLEGE/DS/Code/" && gcc exp7.c -c
************************Binary Search Tree***************************

Enter the number in front the operation to perform it
1.Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 1

Enter the number of elements you want to insert in the binary search tree
10
Enter the value of the element you want to insert in the binary search tree:70
Enter the value of the element you want to insert in the binary search tree:40
Enter the value of the element you want to insert in the binary search tree:30
Enter the value of the element you want to insert in the binary search tree:20
Enter the value of the element you want to insert in the binary search tree:10
Enter the value of the element you want to insert in the binary search tree:3
Enter the value of the element you want to insert in the binary search tree:5
Enter the value of the element you want to insert in the binary search tree:32
Enter the value of the element you want to insert in the binary search tree:56
Enter the value of the element you want to insert in the binary search tree:24

Enter the number in front the operation to perform it
1.Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 4
The elements in th tree are:70 40 30 20 10 3 5 24 32 56
```



```
Enter the number in front the operation to perform it
1.Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 5
The elements in th tree are:3   5       10      20      24      30      32      40      56      70
Enter the number in front the operation to perform it
1.Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 6
The elements in th tree are:5   3       10      24      20      32      30      56      40      70
Enter the number in front the operation to perform it
1.Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 7
Enter the value of the element you want to search in the binary search tree:10
The element is present in the tree
```



```
Enter the number in front the operation to perform it
1.Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 3
Enter the value of the element you want to delete from the binary search tree:60

 The value to be deleted is not present in the tree
Enter the number in front the operation to perform it
1.Create a binary search tree
2. Insert element
3. Delete element
4. Preorder traversal
5. Inorder traversal
6. Postorder traversal
7. Search element
8. Exit
Enter your choice: 8

Exiting the program%
pargat@Router Code %
```

Conclusion:-

Hence, we successfully implemented BST & Binary tree traversal techniques.

PostLab Questions:

1) Illustrate 2 Applications of Trees.

1. General Tree
If no constraint is placed on the tree's hierarchy, a tree is called a general tree. Every node may have infinite numbers of children in General Tree. The tree is the super-set of all other trees.

2. Binary Tree
The binary tree is the kind of tree in which most two children can be found for each parent. The kids are known as the left kid and right kid. This is more popular than most other trees. When certain constraints and characteristics are applied in a Binary tree, a number of others such as AVL tree, BST (Binary Search Tree), RBT tree, etc. are also used. When we move forward, we will explain all these styles in detail.

3. Binary Search Tree
Binary Search Tree (BST) is a binary tree extension with several optional restrictions. The left child value of a node should in BST be less than or equal to the parent value, and the right child value should always be greater than or equal to the parent's value. This Binary Search Tree property makes it ideal for search operations since we can accurately determine at each node whether the value is in the left or right sub-tree. This is why the Search Tree is named.

4. AVL Tree

AVL tree is a binary search tree self-balancing. On behalf of the inventors Adelson-Velshi and Landis, the name AVL is given. This was the first tree that balanced

dynamically. A balancing factor is allocated for each node in the AVL tree, based on whether the tree is balanced or not. The height of the node kids is at most 1. AVL vine. In the AVL tree, the correct balance factor is 1, 0 and -1. If the tree has a new node, it will be rotated to ensure that it is balanced. It will then be rotated. Common operations such as viewing, insertion, and removal take O(log n) time in the AVL tree. It is mostly applied when working with Lookup operations.

5. Red-Black Tree
Another kind of auto-balancing tree is red-black. According to the red-black tree's properties, the red-black name is given because the Red-black tree has either red or Black painted on each node. It maintains the balance of the forest. Even though this tree is not fully balanced, the searching operation only takes O (log n) time. When the new

nodes are added in Red-Black Tree, nodes will be rotated to maintain the Red-Black Tree's properties.

6. N-ary Tree
The maximum number of children in this type of tree with a node is N. A binary tree is a two-year tree, as at most 2 children in every binary tree node. A complete N-ary tree is a tree where kids of a node either are 0 or N.

2) Compare and Contrast between B Tree and B+ Tree?

A tree is a non-linear data structure that consists of a root node and potentially many levels of additional nodes that form a hierarchy. A tree can be empty with no nodes called the null or empty tree or a tree is a structure consisting of one node called the root and one or more subtrees.
A tree contains following operations.
Insert: - To add a new node to the tree.
Search: - To find if a node is present in the tree.
Delete: - To remove a given node from the tree.