K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Batch: **A2** **Roll No.:** **16010122041**

Experiment / assignment / tutorial No. 8

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

In this root is traversed prior to the left and right subtree.

```
Function preorder(root)
1) Start
2) If root is not null then
    a) display the data in the root
    b) call preorder with left pointer of root
    c) call preoder with right pointer of root.
3) Stop.
```

Postorder Traversal of BST

In this root is traversed after the left and right subtree.

```
Function postorder(root)
1) Start
2) If root is not null then
   a) call postorder with left pointer of root
   b) call preoder with right pointer of root.
   c) display the data in the root
3) Stop.
```

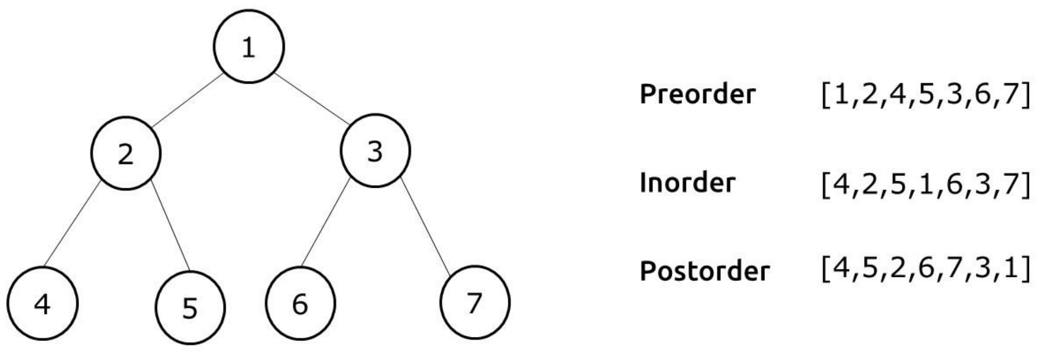
Inorder Traversal of BST

In this root is traversed between the left and right sub tree.

```
Function inorder(root)
1) Start
2) If root is not null then
   b) call inorder with left pointer of root
   c) display the data in the root
   d) call inoder with right pointer of root.
3) Stop.
```

Diagram for :

Preorder Traversal of BST
Postorder Traversal of BST
Inorder Traversal of BST



Algorithm for Implementation of BST & Binary tree traversal techniques:

Implementation Details:

1) Enlist all the Steps followed and various options explored.

Assumptions made for Input:

1. User does not enter a negative number
2. The values of all nodes are integers and |value| <= INTMAX

Built-In Functions Used:

Struct

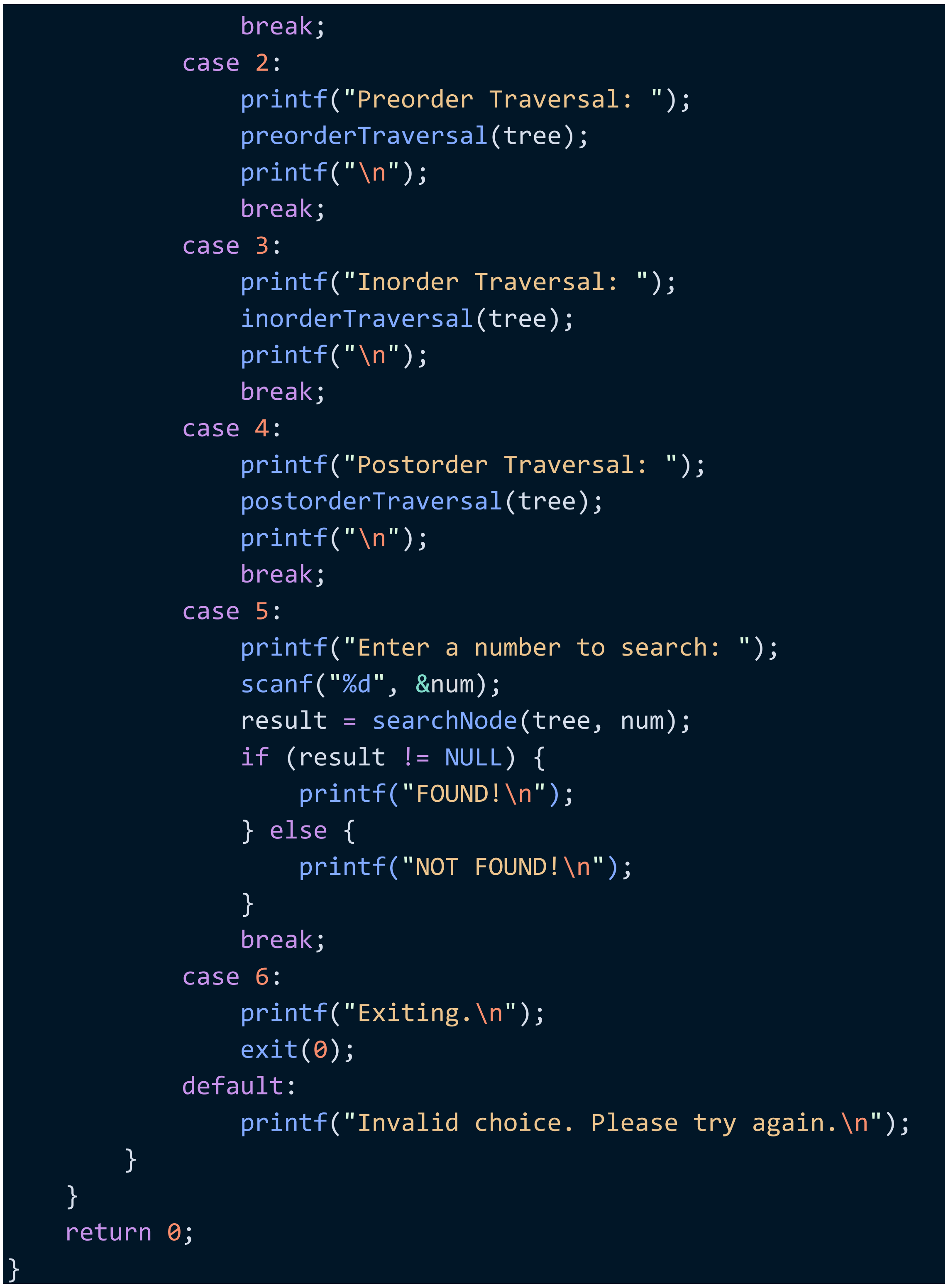
Malloc()

Scanf()

Printf()

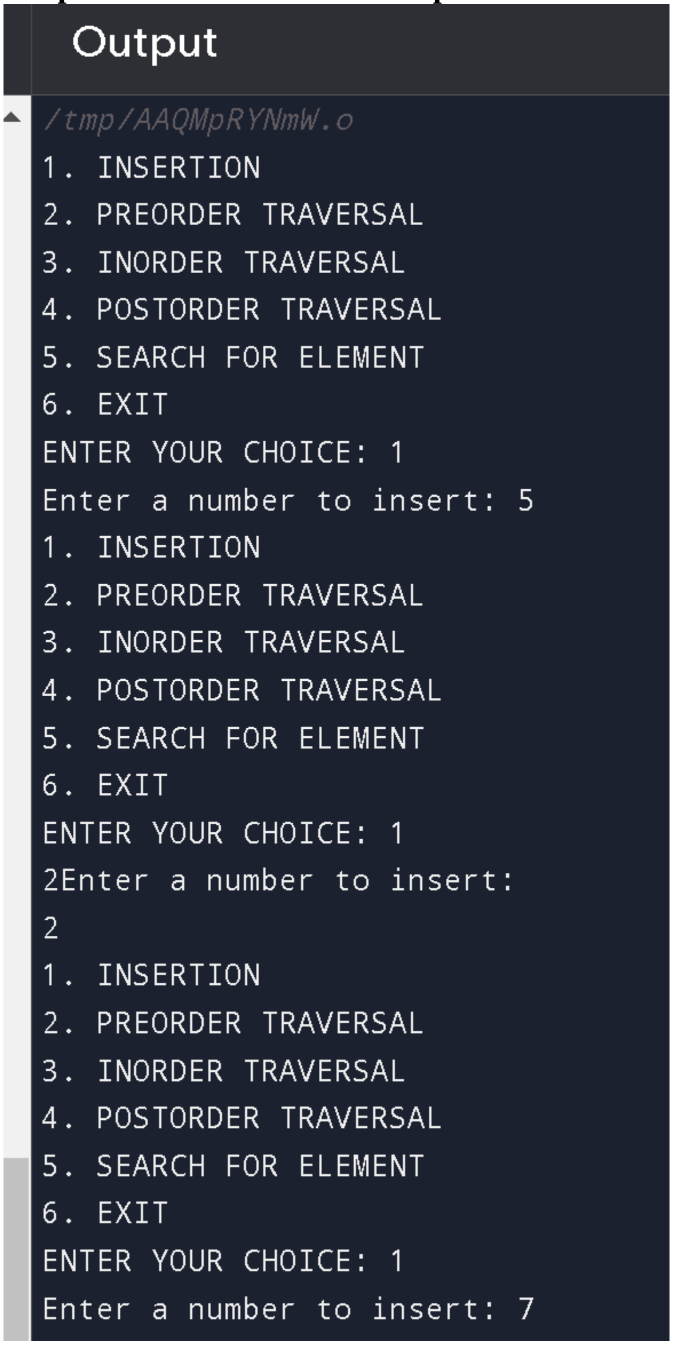
Program source code for Implementation of BST & Binary tree traversal techniques :

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

struct node {
    int num;
    struct node *left;
    struct node *right;
};

struct node *createNode(int num)
{
    struct node *newNode = (struct node *)malloc(sizeof(struct 
node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newNode->num = num;
    newNode->left = newNode->right = NULL;
    return newNode;
}

struct node *insertNode(struct node *root, int num)
{
    if (root == NULL) {
        return createNode(num);
    }
    if (num < root->num) {
        root->left = insertNode(root->left, num);
    } else if (num > root->num) {
        root->right = insertNode(root->right, num);
    }
    return root;
}

void preorderTraversal(struct node *root)
{
    if (root != NULL) {
        printf("%d ", root->num);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
    }
}

void inorderTraversal(struct node *root)
{
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->num);
        inorderTraversal(root->right);
    }
}

void postorderTraversal(struct node *root)
{
    if (root != NULL) {
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->num);
    }
}

struct node *searchNode(struct node *root, int num)
{
    if (root == NULL || root->num == num) {
        return root;
    }
    if (num < root->num) {
        return searchNode(root->left, num);
    }
    return searchNode(root->right, num);
}

int main()
{
    int choice, num;
    struct node *tree;
    struct node *result;
    choice=0;

    while (choice!=6) {
        printf("1. INSERTION\n");
        printf("2. PREORDER TRAVERSAL\n");
        printf("3. INORDER TRAVERSAL\n");
        printf("4. POSTORDER TRAVERSAL\n");
        printf("5. SEARCH FOR ELEMENT\n");
        printf("6. EXIT\n");
        printf("ENTER YOUR CHOICE: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter a number to insert: ");
                scanf("%d", &num);
                tree = insertNode(tree, num);
```



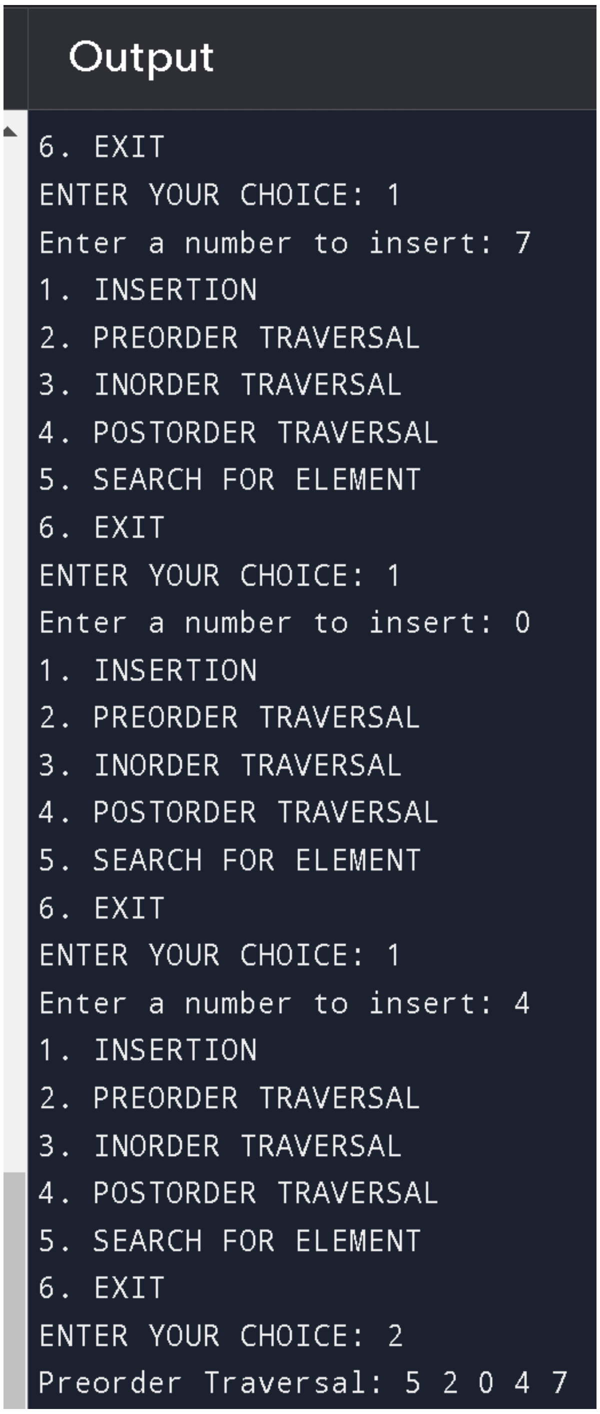
```
                break;
            case 2:
                printf("Preorder Traversal: ");
                preorderTraversal(tree);
                printf("\n");
                break;
            case 3:
                printf("Inorder Traversal: ");
                inorderTraversal(tree);
                printf("\n");
                break;
            case 4:
                printf("Postorder Traversal: ");
                postorderTraversal(tree);
                printf("\n");
                break;
            case 5:
                printf("Enter a number to search: ");
                scanf("%d", &num);
                result = searchNode(tree, num);
                if (result != NULL) {
                    printf("FOUND!\n");
                } else {
                    printf("NOT FOUND!\n");
                }
                break;
            case 6:
                printf("Exiting.\n");
                exit(0);
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
```

Output Screenshots for Each Operation:

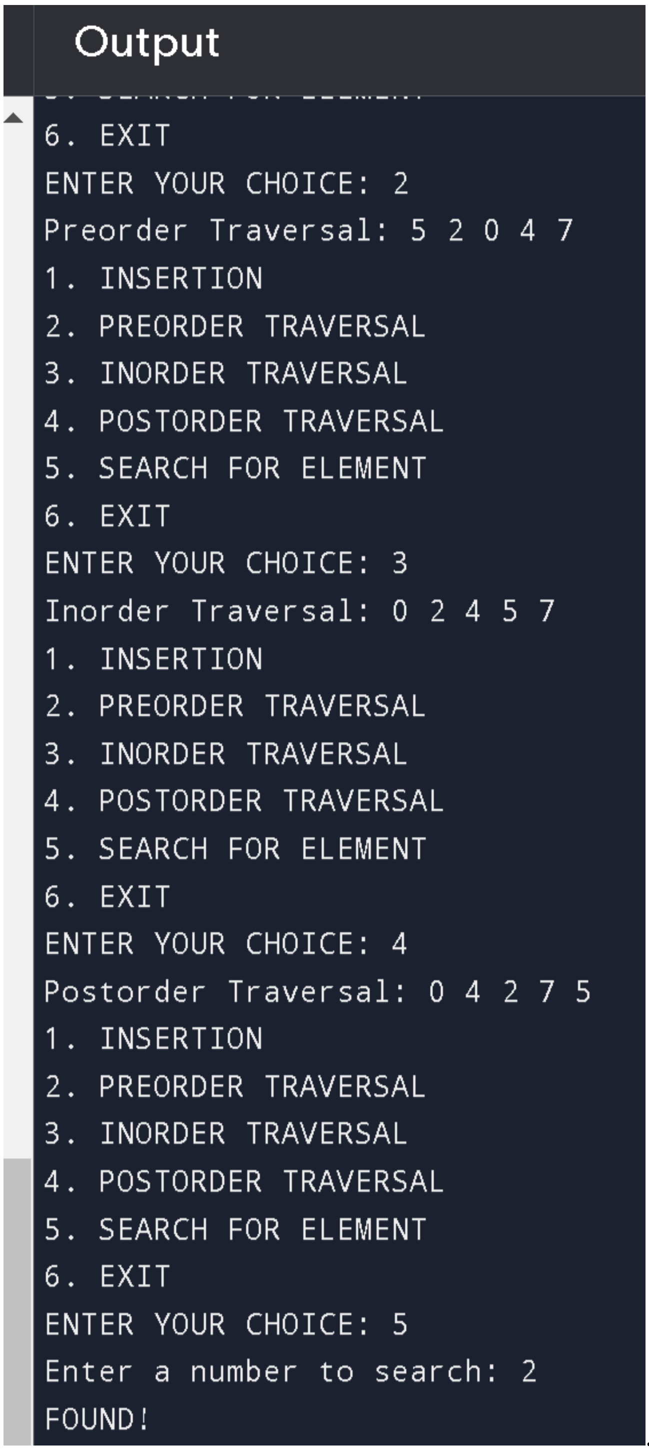


Output

```
/tmp/AAQMpRYNmW.o
1. INSERTION
2. PREORDER TRAVERSAL
3. INORDER TRAVERSAL
4. POSTORDER TRAVERSAL
5. SEARCH FOR ELEMENT
6. EXIT
ENTER YOUR CHOICE: 1
Enter a number to insert: 5
1. INSERTION
2. PREORDER TRAVERSAL
3. INORDER TRAVERSAL
4. POSTORDER TRAVERSAL
5. SEARCH FOR ELEMENT
6. EXIT
ENTER YOUR CHOICE: 1
2Enter a number to insert:
2
1. INSERTION
2. PREORDER TRAVERSAL
3. INORDER TRAVERSAL
4. POSTORDER TRAVERSAL
5. SEARCH FOR ELEMENT
6. EXIT
ENTER YOUR CHOICE: 1
Enter a number to insert: 7
```



```
6. EXIT
ENTER YOUR CHOICE: 1
Enter a number to insert: 7
1. INSERTION
2. PREORDER TRAVERSAL
3. INORDER TRAVERSAL
4. POSTORDER TRAVERSAL
5. SEARCH FOR ELEMENT
6. EXIT
ENTER YOUR CHOICE: 1
Enter a number to insert: 0
1. INSERTION
2. PREORDER TRAVERSAL
3. INORDER TRAVERSAL
4. POSTORDER TRAVERSAL
5. SEARCH FOR ELEMENT
6. EXIT
ENTER YOUR CHOICE: 1
Enter a number to insert: 4
1. INSERTION
2. PREORDER TRAVERSAL
3. INORDER TRAVERSAL
4. POSTORDER TRAVERSAL
5. SEARCH FOR ELEMENT
6. EXIT
ENTER YOUR CHOICE: 2
Preorder Traversal: 5 2 0 4 7
```



Output

```
6. EXIT
ENTER YOUR CHOICE: 2
Preorder Traversal: 5 2 0 4 7
1. INSERTION
2. PREORDER TRAVERSAL
3. INORDER TRAVERSAL
4. POSTORDER TRAVERSAL
5. SEARCH FOR ELEMENT
6. EXIT
ENTER YOUR CHOICE: 3
Inorder Traversal: 0 2 4 5 7
1. INSERTION
2. PREORDER TRAVERSAL
3. INORDER TRAVERSAL
4. POSTORDER TRAVERSAL
5. SEARCH FOR ELEMENT
6. EXIT
ENTER YOUR CHOICE: 4
Postorder Traversal: 0 4 2 7 5
1. INSERTION
2. PREORDER TRAVERSAL
3. INORDER TRAVERSAL
4. POSTORDER TRAVERSAL
5. SEARCH FOR ELEMENT
6. EXIT
ENTER YOUR CHOICE: 5
Enter a number to search: 2
FOUND!
```

.

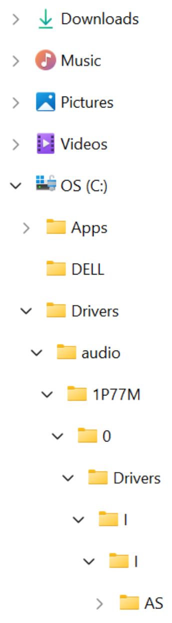
Conclusion:-

We learnt about Binary Search Tree and its implementation.

PostLab Questions:

1) Illustrate 2 Applications of Trees.

a. File System
The files and folders that you see in your windows explorer are stored in the tree format. In the below image you can say that 'My Computer' is the root; Local Disk (C), Local Disk (D), and Local Disk (E) are basically the parent nodes and files inside them are leaf nodes. This allows faster traversal of the nodes while jumping from one node to another.



b. Webpage Layout
The layout of a webpage is designed in the tree structure. In the below diagram, the homepage or index page is our root node, main sections/ site index are their child nodes, which again are parents to multiple other child nodes (subsections).

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

B tree	B+ tree
In the B tree, all the keys and records are stored in both internal as well as leaf nodes.	In the B+ tree, keys are the indexes stored in the internal nodes and records are stored in the leaf nodes.
In B tree, keys cannot be repeatedly stored, which means that there is no duplication of keys or records.	In the B+ tree, there can be redundancy in the occurrence of the keys. In this case, the records are stored in the leaf nodes, whereas the keys are stored in the internal nodes, so redundant keys

	can be present in the internal nodes.
In the B tree, leaf nodes are not linked to each other.	In B+ tree, the leaf nodes are linked to each other to provide the sequential access.
In B tree, searching is not very efficient because the records are either stored in leaf or internal nodes.	In B+ tree, searching is very efficient or quicker because all the records are stored in the leaf nodes.
Deletion of internal nodes is very slow and a time-consuming process as we need to consider the child of the deleted key also.	Deletion in B+ tree is very fast because all the records are stored in the leaf nodes so we do not have to consider the child of the node.
In B tree, sequential access is not possible.	In the B+ tree, all the leaf nodes are connected to each other through a pointer, so sequential access is possible.
In B tree, the greater number of splitting operations are performed due to which height increases compared to width.	B+ tree has more width as compared to height.
In Btree, each node has atleast two branches and each node contains some records, so we do not need to traverse till the leaf nodes to get the data.	In B+ tree, internal nodes contain only pointers and leaf nodes contain records. All the leaf nodes are at the same level, so we need to traverse till the leaf nodes to get the data.
The root node contains atleast 2 to m children where m is the order of the tree.	The root node contains atleast 2 to m children where m is the order of the tree.