Experiment No. 04

Aim :- Implementation of Binary Tree and its Traversal for real-world application.

Objectives:-

1. To learn fundamentals and implementation of Binary Tree.
2. To develop an ability to design and analyze algorithms using tree data structure.

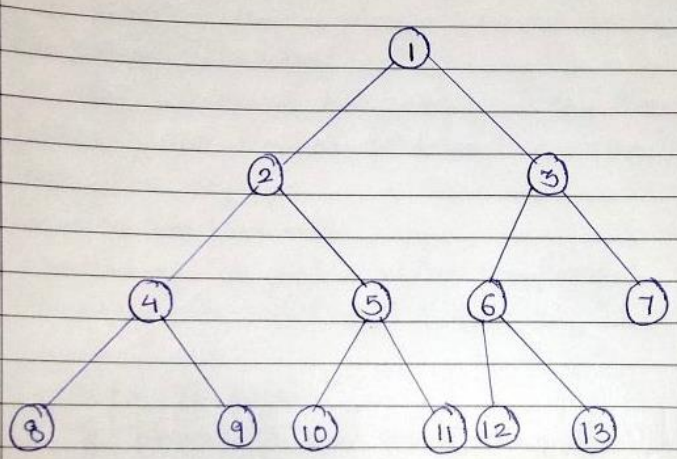
Theory:-

A Binary Tree is a data structure that is defined as a collection of elements called nodes. In a binary tree, the topmost element is called the root node, and each node has 0, 1 or at the most 2 children. A node that has zero children is called a leaf node or a terminal node. Every node contains a data element, a left pointer which points to the right child. The root element is pointed by a 'root' pointer.

Terminology:

- Parent: If N is any node in T that has left successor S_1 and right successor S_2, then N is called the parent of S_1 & S_2.
- Level number: Every node in the binary tree is assigned to a to a level number.
- Degree of a Node: It is equal to the number of children that a node has.
- Sibling: All nodes that are at the same level and share the same parent are called siblings.

- Leaf node: A node that has no children
- Similar binary trees: Two binary trees are said to be similar if both these trees have the same structure.
- Edge :- It is the line connecting a node N to any of its successors.
- Path :- A sequence of consecutive edges.
- Depth :- The depth of a node is given as the length of the Path from the root to the node.
- Height of a tree :- It is the total number of nodes on the path from the root node to the deepest node in the tree.



Operations :-

1. Searching → Find the location of some specific elements in a binary tree.

2. Insertion → Adding a new element to the tree at the appropriate location.

3. Deletion → Deleting some specific node from a binary tree.

4. Traversing → Process of visiting

Tree traversal and its types:

Traversing a binary tree is the process of visiting each node in the tree exactly once in a systematic way unlike linear data structures in which the elements are traversed sequentially, tree is a non-linear data structure in which the elements can traversed in many different way.

1. Pre-order Traversal.

To traverse a non-empty binary tree in pre-order, the following operations are performed recursively at each node the algorithm ways.

- Visiting the root node
- Traversing the left sub-tree and finally.
- Traversing the right sub-tree.

2. In-order Traversal.

To traverse a non-empty binary tree in in-order the following operation are performed recursively at each node. The algorithm work by:

- Traversing the left sub-tree
- Visiting the root node and finally.
- Traversing the right subtree.

5. Post-order traversal :-

To traverse a non empty binary tree in post-order, the following operations are performed recursively at each node. The algorithm works by

- Traversing the left sub-tree.
- Traversing the right sub-tree & finally
- Visiting the root node.

Algorithms :-

→ Searching for a given value.

Step 1 :- If Tree → DATA = VAL OR TREE = NULL
Return TREE
ELSE
If VAL < TREE → DATA
Return Search Element (TREE → LEFT VAL)
ELSE
Return Search Element (TREE → RIGHT, VALUE)
[END. of IF]
[END OF IF].

Step 2 :- END

→ Insertion :- INSERT (TREE, VAL).

Step 1 :- IF TREE = NULL
Allocate memory for Tree.
SET TREE → DATA = VAL
SET TREE → LEFT = TREE → RIGHT = NULL.
ELSE
IF VAL < TREE → DATA.
INSERT (TREE → LEFT, VAL.)

```
ELSE
        IF VAL < TREE -> DATA
            INSERT (TREE -> RIGHT, VAL)
        [END OF IF]
    [END OF IF].
Step 2: END
```

→ DELETION

```
Delete (TREE, VAL).
Step 1: If TREE = NULL.
    Write "VAL not found in the tree".
    Else if VAL < TREE -> DATA
      Delete (TREE -> LEFT, VAL)
    Else if VAL > TREE -> DATA.
      Delete (TREE -> RIGHT, VAL).
    Else if TREE -> LEFT AND TREE -> RIGHT
      SET TEMP = Find largest Node (TREE -> LEFT)
      SET TREE -> DATA = TEMP -> DATA
      DELETE (TREE -> LEFT, TEMP -> DATA.
    ELSE
      SET TEMP = TREE
    IF TREE -> LEFT = NULL AND TREE -> RIGHT = NULL.
      SET TREE = NULL
    Else IF TREE -> LEFT != NULL
      SET TREE = TREE -> LEFT
    Else
      SET TREE = TREE -> RIGHT.
    [END OF IF]
    FREE TEMP
    [END OF IF]
```

Step 2:- END.

→ Pre-order Traversal.
Step 1: Repeat Step 2 to 4 while TREE != NULL.
Step 2: Write TREE → DATA.
Step 3: PREORDER (TREE → LEFT)
Step 4: PREORDER (TREE → RIGHT).
END OF LOOP
Step 5: END

→ Inorder Traversal.
Step 1: Repeat steps 2 to 4 while TREE != NULL
Step 2: INORDER (TREE → LEFT)
Step 3: Write TREE → RIGHT
Step 4: INORDER (TREE → RIGHT).
[END OF LOOP]
Step 5: END

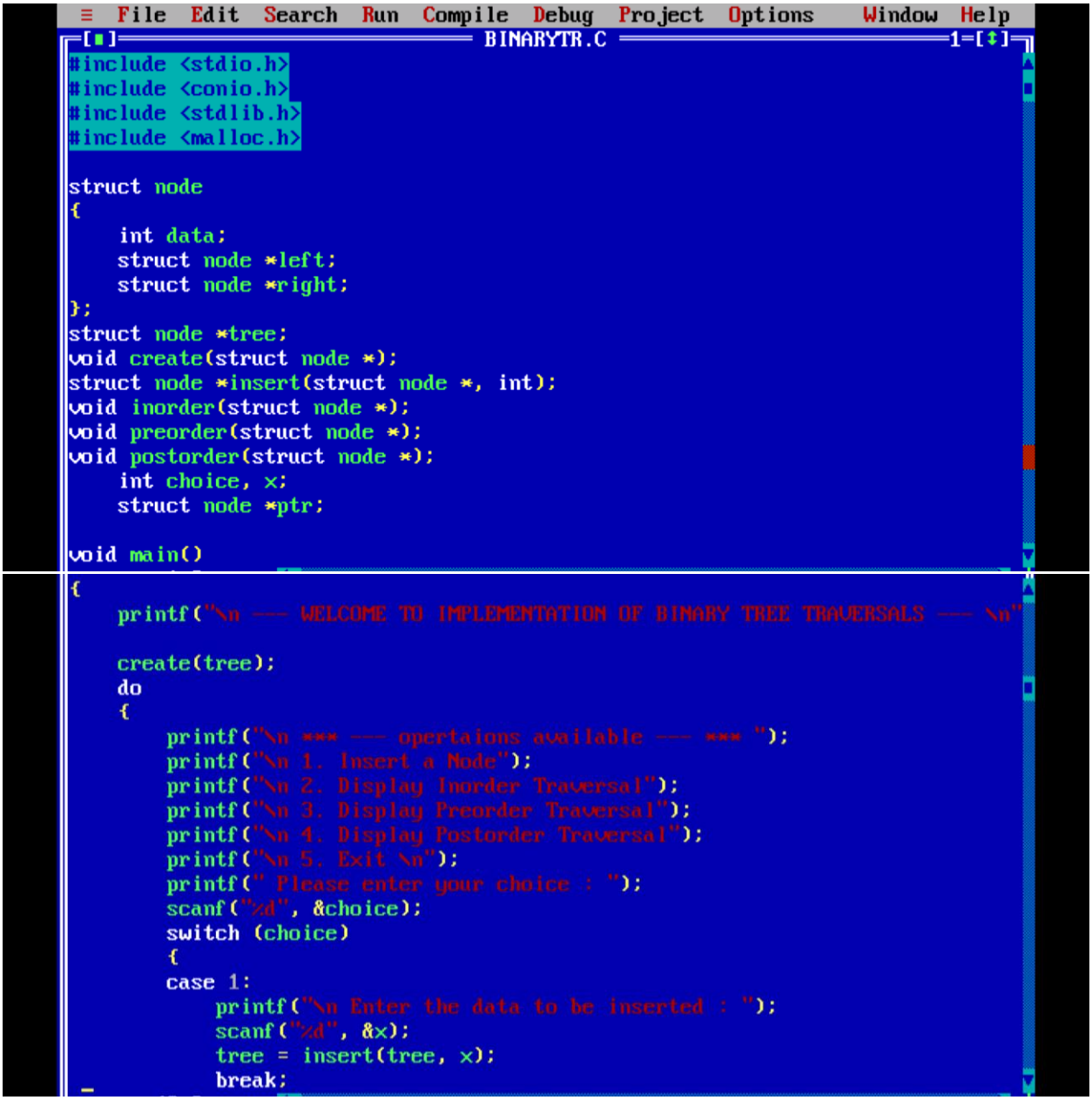
→ Post-order Traversal
Step 1: Repeat steps 2 to 4 while TREE != NULL.
Step 2: POSTORDER (TREE → LEFT)
Step 3: POSTORDER (TREE → RIGHT)
Step 4: Write TREE → DATA.
[END OF LOOP]
Step 5: END.

Example:-

- Routing Tables: A routing table is used to link routers in a network.
- Trees are used in file system directories.
- Trees are widely used for information storage and retrieval in symbol tables.

Conclusion: Thus, we understand the concept of binary trees, their operations including traversal and its various types and also learn its implementation.

Outcome: Implement tree data structure for real-world application.

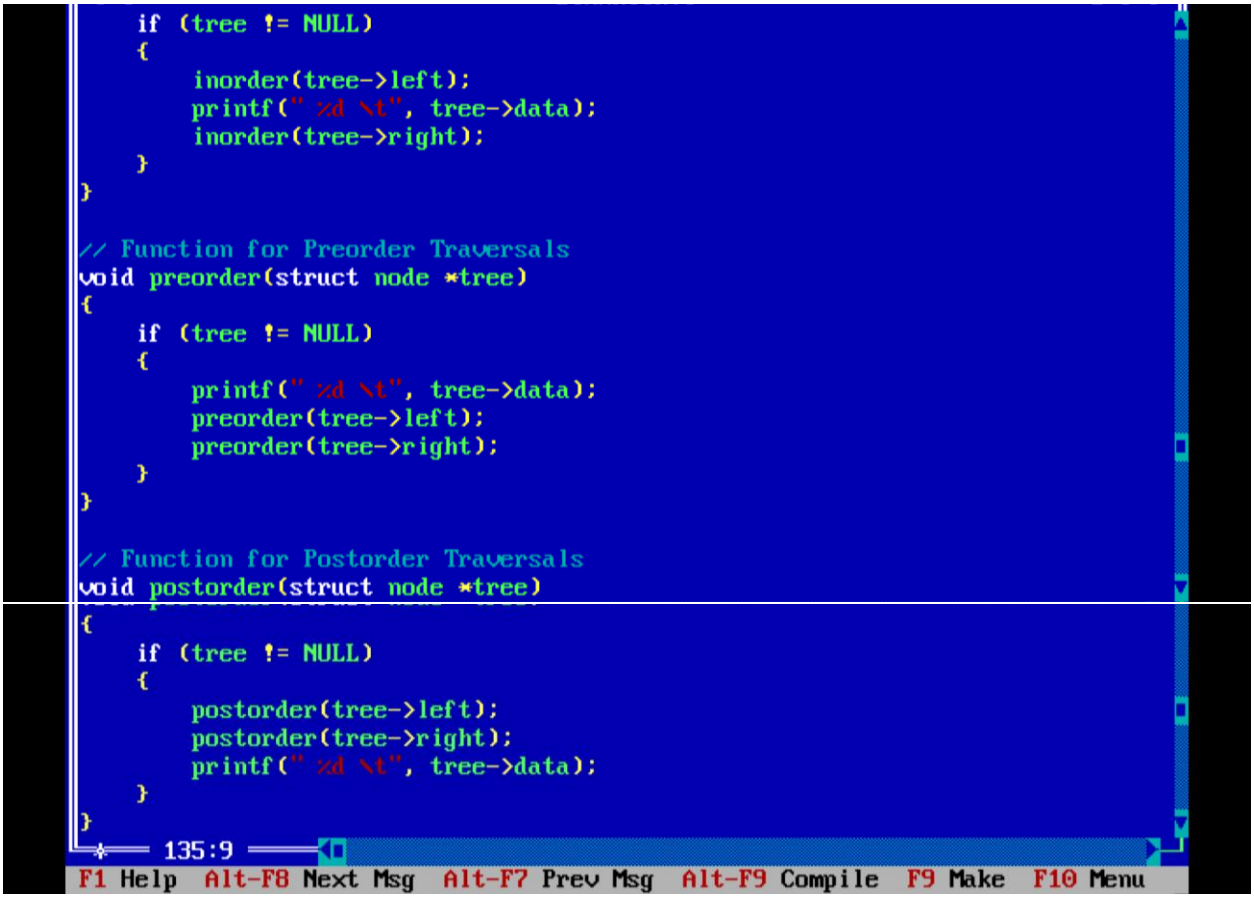


```
 ≡  File  Edit  Search  Run  Compile  Debug  Project  Options    Window  Help
[■]═══════════════════ BINARYTR.C ═══════════════════1═[↕]═
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <malloc.h>

struct node
{
    int data;
    struct node *left;
    struct node *right;
};
struct node *tree;
void create(struct node *);
struct node *insert(struct node *, int);
void inorder(struct node *);
void preorder(struct node *);
void postorder(struct node *);
    int choice, x;
    struct node *ptr;

void main()
{
    printf("\n ---- WELCOME TO IMPLEMENTATION OF BINARY TREE TRAVERSALS ---- \n"

    create(tree);
    do
    {
        printf("\n *** ---- opertaions available ---- *** ");
        printf("\n 1. Insert a Node");
        printf("\n 2. Display Inorder Traversal");
        printf("\n 3. Display Preorder Traversal");
        printf("\n 4. Display Postorder Traversal");
        printf("\n 5. Exit \n");
        printf(" Please enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("\n Enter the data to be inserted : ");
            scanf("%d", &x);
            tree = insert(tree, x);
            break;
```

```
        case 2:
            printf("\n Elements in the inorder traversala are : ");
            inorder(tree);
            printf("\n");
            break;
        case 3:
            printf("\n Elements in the preorder traversala are : ");
            preorder(tree);
            printf("\n");
            break;
        case 4:
            printf("\n Elements in the postorder traversala are : ");
            postorder(tree);
            printf("\n");
            break;
        default:
            printf("\n Please enter a valid option 1, 2, 3, 4.");
            break;
        }
    } while (choice != 5);
}
```

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

// Function for inserting a new node
struct node *insert(struct node *tree, int x)
{
    struct node *p, *temp, *root;
    p = (struct node *)malloc(sizeof(struct node));
    p->data = x;
    p->left = NULL;
    p->right = NULL;
    if (tree == NULL)
    {
        tree = p;
        tree->left = NULL;
        tree->right = NULL;
    }
    else
    {
```

```
        root = NULL;
        temp = tree;
        while (temp != NULL)
        {
            root = temp;
            if (x < temp->data)
                temp = temp->left;
            else
                temp = temp->right;
        }
        if (x < root->data)
            root->left = p;
        else
            root->right = p;
    }
    return tree;
}

// Function for Inorder Traversals
void inorder(struct node *tree)
{
    if (tree != NULL)
    {
        inorder(tree->left);
        printf(" %d \t", tree->data);
        inorder(tree->right);
    }
}

// Function for Preorder Traversals
void preorder(struct node *tree)
{
    if (tree != NULL)
    {
        printf(" %d \t", tree->data);
        preorder(tree->left);
        preorder(tree->right);
    }
}

// Function for Postorder Traversals
void postorder(struct node *tree)
```


if (tree != NULL)
{
inorder(tree->left);
printf(" %d \t", tree->data);
inorder(tree->right);
}
}
// Function for Preorder Traversals
void preorder(struct node *tree)
{
if (tree != NULL)
{
printf(" %d \t", tree->data);
preorder(tree->left);
preorder(tree->right);
}
}
// Function for Postorder Traversals
void postorder(struct node *tree)
{
if (tree != NULL)
{
postorder(tree->left);
postorder(tree->right);
printf(" %d \t", tree->data);
}
}
135:9
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu

```
--- WELCOME TO IMPLEMENTATION OF BINARY TREE TRAVERSALS ---

*** --- opertaions available --- ***
1. Insert a Node
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
Please enter your choice : 1

Enter the data to be inserted : 77

*** --- opertaions available --- ***
1. Insert a Node
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
Please enter your choice : 1

Enter the data to be inserted : 84_
```

```
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
Please enter your choice : 1

Enter the data to be inserted : 99

*** --- opertaions available --- ***
1. Insert a Node
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
Please enter your choice : 1

Enter the data to be inserted : 22

*** --- opertaions available --- ***
1. Insert a Node
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
```

```
Please enter your choice : 1

Enter the data to be inserted : 55

*** --- opertaions available --- ***
1. Insert a Node
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
Please enter your choice : 2

Elements in the inorder traversala are :  22    55      77      84      99
```

```
*** --- opertaions available --- ***
1. Insert a Node
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
Please enter your choice :
3

Elements in the preorder traversala are :  77   22      55      84      99


*** --- opertaions available --- ***
1. Insert a Node
2. Display Inorder Traversal
3. Display Preorder Traversal
4. Display Postorder Traversal
5. Exit
Please enter your choice : _
```

```
Please enter your choice :    4

Elements in the postorder traversala are :  55          22      99      84
77
```