## 14. IMPLEMENTATION OF TREE TRAVERSAL

### Preamble

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree:

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

### Steps - In-order Traversal

In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself. Until all nodes are traversed −

Step 1 − Recursively traverse left subtree.

Step 2 − Visit root node.

Step 3 − Recursively traverse right subtree.

### Steps - Pre-order Traversal

In this traversal method, the root node is visited first, then the left subtree and finally the right subtree. Until all nodes are traversed −

Step 1 − Visit root node.

Step 2 − Recursively traverse left subtree.

Step 3 − Recursively traverse right subtree.

### Steps - Post-order Traversal

In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node. Until all nodes are traversed:

Step 1 − Recursively traverse left subtree.

Step 2 − Recursively traverse right subtree.

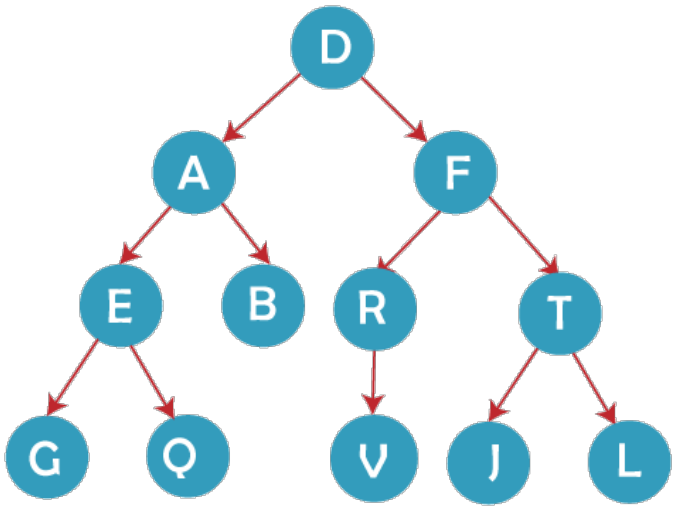
Step 3 − Visit root node.

**Implementation in C**

```
#include <stdio.h>

int max_node = 15;

char tree[] = {'\0', 'D', 'A', 'F', 'E', 'B', 'R', 'T', 'G', 'Q', '\0', '\0',
'V', '\0', 'J', 'L'};

int get_right_child(int index)

{

      if (tree[index] != '\0' && ((2 * index) + 1) <= max_node)

            return (2 * index) + 1;

      return -1;

}


int get_left_child(int index)

{

      if (tree[index] != '\0' && (2 * index) <= max_node)

            return 2 * index;

      return -1;

}


void preorder(int index)

{

      if (index > 0 && tree[index] != '\0')

      {

            printf(" %c ", tree[index]);

            preorder(get_left_child(index));

            preorder(get_right_child(index));

      }

}


void postorder(int index)

{

      if (index > 0 && tree[index] != '\0')
```

```
	{
		postorder(get_left_child(index));
		postorder(get_right_child(index));
		printf(" %c ", tree[index]);
	}
}

void inorder(int index)
{
	if (index > 0 && tree[index] != '\0')
	{
		inorder(get_left_child(index));
		printf(" %c ", tree[index]);
		inorder(get_right_child(index));
	}
}

int main()
{
	printf("Preorder:\n");
	preorder(1);
	printf("\nPostorder:\n");
	postorder(1);
	printf("\nInorder:\n");
	inorder(1);
	printf("\n");
	return 0;
}
```

**Sample Input and Output**




Preorder:
D A E G Q B F R V T J L
Postorder:
G Q E B A V R J L T F D
Inorder:
G E Q A B D V R F J T L