**Lab 8**

**Binary Tree Traversals (iterative)**

**Introduction:**

A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called the left sub tree and the right sub tree.

**Types & Implementation**

* Full binary tree: Every node other than **leaf** nodes has 2 child nodes.
* Complete binary tree: All levels are filled except possibly the last one, and all nodes are filled in as far left as possible.
* Perfect binary tree: All nodes have two children and all **leaves** are at the same level.

**Binary tree traversals:**

Since a binary tree is a non-linear data structure, the problem is how to traverse it so we visit each node exactly once. Following are the basic traversal methodologies in which we traverse the left sub tree before the right sub tree.



**Preorder traversal:**

To traverse a binary tree in Preorder (VLR) manner following operations are carried-out

**(i)** Visit the root,

**(ii)** Traverse the left subtree, and

**(iii)** Traverse the right subtree. Therefore, the Preorder traversal of the tree will output: **+\*\*/ABCDE**

**In order traversal:**

To traverse a binary tree in Inorder (LVR), following operations are carried-out

**(i)** Traverse the left subtree

**(ii)** Visit the root, and

**(iii)** Traverse the right subtree. Therefore, the Inorder traversal of the tree will output: **A/B\*C\*D+E**

**Post order traversal:**

To traverse a binary tree in Post order (LRV), following operations are carried out

**(i)** Traverse all the left subtree

**(ii)** Traverse the right subtree and

**(iii)** Visit the root. Therefore, the Post order traversal of the tree will output: **AB/C\*D\*E+**

**Level order traversal:**

To traverse a binary tree in level order manner, we traverse each level of the tree from left to right starting from level 1 and moving down. Therefore, the Level order traversal of the tree will output: +\*E\*D/CAB

**OBJECTIVE:**

* The objective of this experiment is to implement the basic binary tree traversals iteratively.
* The objective of this experiment is to build a binary tree and then implement the basic binary tree traversals recursively.
* Learn how to implement **binary search trees** and **binary** heaps, and are used for efficient searching and sorting.

**APPLICATION:**

* **Binary search trees** allow **binary** search for fast lookup, addition and removal of data items, and can be **used to** implement dynamic sets and lookup tables.
* **Binary Search Tree** - Used in many search applications where data is constantly entering/leaving, such as the map and set objects in many languages' libraries.
* **Binary Space Partition** - Used in almost every 3D video game to determine what objects need to be rendered.

**ISSUE:**

**CONCLUSION:**

* Binary search trees are a very powerful (but not perfect) data **structure** to have in your programming tool belt. If done right, handling large amounts of sorted data becomes easier and quicker.