**Name: Sundas Shoukat**

**Roll no. 057**

**Section: BSDS-3A**

**Task 1:**

**DFS with Stack**

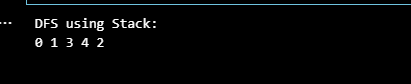
**How it works:**

1. We use a **stack** instead of recursion.
2. First, we put the **starting node** in the stack.
3. Then we repeat these steps until the stack is empty:
   * Remove (pop) the top node from the stack.
   * If it is not visited, we **mark it visited** and **print it**.
   * Then we add its **neighbors** into the stack (so they will be visited later).
4. This way, the stack keeps track of the next nodes, and we go **deep first** like DFS.

### Why we do it this way:

* Normally DFS is written using **recursion**, but recursion internally also uses a **stack**.
* By writing it with our own stack, we can see **how DFS actually works inside**.
* This method is helpful when recursion is **not allowed** or when the graph is **very big** (to avoid recursion depth error).
* It is a clear and simple way to understand **Depth First Search step by step**.

**OUTPUT:**

****

**Task 2:**

**Inorder, Preorder, Postorder Traversals in DFS**

**How it works:**

1. We make a **binary tree** using a simple Node class.

* Each node has a value, a left child, and a right child.

1. Then we create three traversal functions:

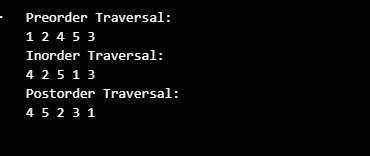
* **Preorder (Root → Left → Right):** Visit the root node first, then the left child, then the right child.
* **Inorder (Left → Root → Right):** Visit the left child first, then the root, and then the right child.
* **Postorder (Left → Right → Root):** Visit the left child first, then the right child, and root at the end.

1. Each function uses **recursion** to go deeper into the tree until all nodes are visited.
2. We call these functions one by one to print the traversal result.

**Why we do it this way:**

* **Preorder** is useful when we want to **copy the tree** or create a **prefix expression**.
* **Inorder** is useful in a **Binary Search Tree (BST)**, because it gives all elements in **sorted order**.
* **Postorder** is useful when we want to **delete or free the tree** safely, or when creating a **postfix expression**.
* These three methods show us **different ways to explore the same tree**, and they are all based on **Depth First Search (DFS)**.
* Learning them helps us understand how **tree data structures** can be processed in different applications.

**OUTPUT:**

****

**🔹 1. Preorder Traversal**

**Definition:**  
Preorder traversal is a tree traversal method where the **root node is visited first**, then the left subtree, and finally the right subtree.

**Order Rule:**  
Root → Left → Right

**Use:**

* Used to **copy a tree**.
* Used to produce **prefix expressions** in mathematics.

**Example:**  
For the tree:

1

/ \

2 3

/ \

4 5

Preorder = **1 2 4 5 3**

🔹 2. Inorder Traversal

**Definition:**  
Inorder traversal is a tree traversal method where the **left subtree is visited first**, then the root, and finally the right subtree.

**Order Rule:**  
Left → Root → Right

**Use:**

* In **Binary Search Trees (BSTs)**, Inorder traversal gives the **sorted order** of elements.

**Example:**  
For the same tree,  
Inorder = **4 2 5 1 3**

**🔹 3. Postorder Traversal**

**Definition:**  
Postorder traversal is a tree traversal method where the **left subtree is visited first**, then the right subtree, and the root is visited last.

**Order Rule:**  
Left → Right → Root

**Use:**

* Used when **deleting a tree** (because we delete children before parent).
* Used to generate **postfix expressions**.

**Example:**  
For the same tree,  
Postorder = **4 5 2 3 1**