**Experiment No 12:**

Perform traversing of a binary tree

**Aim:** A program to perform traversing of a binary tree.

**Theory:**

A tree data structure is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes.

**Traversing Binary Trees:**

There are 3 ways of traversing a binary tree T with root R.

A tree data structure is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes.

Preorder:

1.Process the root R

2.Traverse the left subtree of R in preorder

3.Traverse the right subtree of R in preorder

Inorder:

1.Traverse the left subtree of R in preorder

2.Process the root R

3.Traverse the right subtree of R in preorder

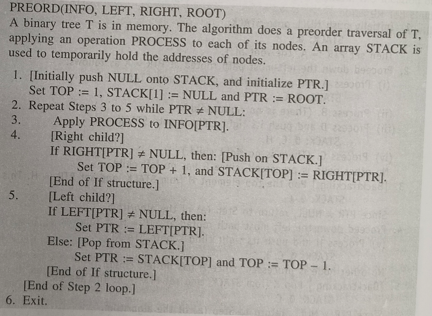
Postorder:

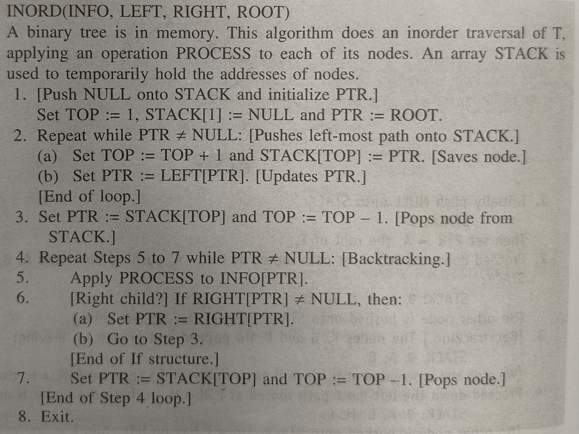
1.Traverse the left subtree of R in preorder

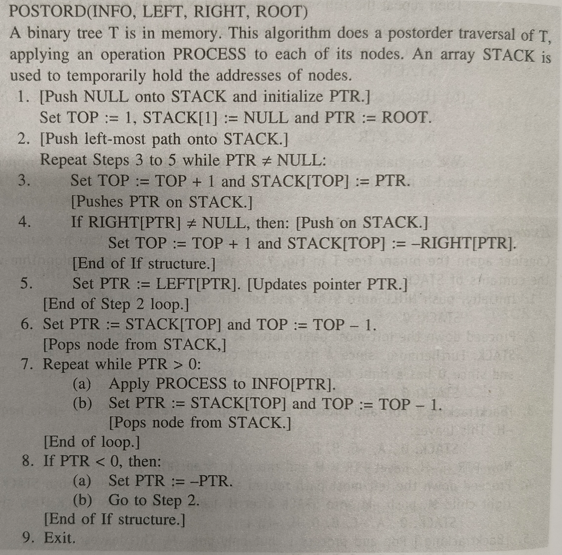
2.Traverse the right subtree of R in preorder

3.Process the root R

**Algorithm:**

****

****

****

**PROGRAM: [Write program to traverse a binary tree]**

 // Tree traversal in C

#include <stdio.h>

#include <stdlib.h>

struct node {

int item;

struct node\* left;

struct node\* right;

};

// Inorder traversal

void inorderTraversal(struct node\* root) {

if (root == NULL) return;

inorderTraversal(root->left);

printf("%d ->", root->item);

inorderTraversal(root->right);

}

// preorderTraversal traversal

void preorderTraversal(struct node\* root) {

if (root == NULL) return;

printf("%d ->", root->item);

preorderTraversal(root->left);

preorderTraversal(root->right);

}

// postorderTraversal traversal

void postorderTraversal(struct node\* root) {

if (root == NULL) return;

postorderTraversal(root->left);

postorderTraversal(root->right);

printf("%d ->", root->item);

}

// Create a new Node

struct node\* createNode(value) {

struct node\* newNode = malloc(sizeof(struct node));

newNode->item = value;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

// Insert on the left of the node

struct node\* insertLeft(struct node\* root, int value) {

root->left = createNode(value);

return root->left;

}

// Insert on the right of the node

struct node\* insertRight(struct node\* root, int value) {

root->right = createNode(value);

return root->right;

}

int main() {

struct node\* root = createNode(1);

insertLeft(root, 12);

insertRight(root, 9);

insertLeft(root->left, 5);

insertRight(root->left, 6);

printf("Inorder traversal \n");

inorderTraversal(root);

printf("\nPreorder traversal \n");

preorderTraversal(root);

printf("\nPostorder traversal \n");

postorderTraversal(root);

}

**OUTPUT:**

**CONCLUSION:**