**NAME:** AGNIV PRAMANICK **SECTION:** A **USN:** 1NT21IS017 **DATE:** 28/02/23

**Q. Write a program to implement binary tree and traverse it using c program.**

**THEORY**

A tree data structure known as a binary tree allows each parent node to have a maximum of two offspring. A binary tree has three components at each node:

• data item

• left child's address

• the right child's address

PROPERTIES:

1. The most nodes are allowed at each rank of I is 2i.
2. The distance from the root node to the limb node that is the longest is considered the tree's height. The tree depicted above is three stories tall. Therefore, (1+2+4+8) = 15 is the greatest number of nodes at height 3. The largest number of nodes at height h is typically (20 + 21 + 22 +....2h) = 2h+1 -1.
3. H+1 is the smallest number of nodes that can exist at height h.
4. The height of the tree would be at its greatest if the number of nodes was at its lowest. In contrast, the height of the tree would be the smallest if the number of branches was maximum.

//////////////////////////////////////////////////////////////////////////////////////

**ALGORITHM**

**Binary Tree Insertion Algorithm**

1. If the tree is empty, create a new node and make it the root of the tree
2. Otherwise, traverse the tree starting from the root
3. If the data to be inserted is less than the data of the current node, move to its left child
4. If the data to be inserted is greater than the data of the current node, move to its right child
5. Repeat steps 3 and 4 until a leaf node is reached
6. Create a new node with the given data and make it a child of the leaf node reached in step 5

**In-order Traversal Algorithm**

1. Traverse the left subtree of the root recursively until a leaf node is reached
2. Print the data of the root
3. Traverse the right subtree of the root recursively until a leaf node is reached

**Pre-order Traversal Algorithm**

1. Print the data of the root
2. Traverse the left subtree of the root recursively until a leaf node is reached
3. Traverse the right subtree of the root recursively until a leaf node is reached

**Post-order Traversal Algorithm**

1. Traverse the left subtree of the root recursively until a leaf node is reached
2. Traverse the right subtree of the root recursively until a leaf node is reached
3. Print the data of the root

/////////////////////////////////////////////////////////////////////////////////////////////////

**CODE**

#include <stdio.h>

#include <stdlib.h>

// Define the structure for each node in the binary tree

struct Node {

int data; // Data of the node

struct Node\* left; // Pointer to the left child of the node

struct Node\* right; // Pointer to the right child of the node

};

// Create a new node with the given data and return a pointer to it

struct Node\* createNode(int data) {

struct Node\* node = (struct Node\*)malloc(sizeof(struct Node)); // Allocate memory for the new node

node->data = data; // Set the data of the node

node->left = NULL; // Initialize the left child to NULL

node->right = NULL; // Initialize the right child to NULL

return node; // Return the pointer to the new node

}

// Traverse the binary tree in inorder (left subtree, root, right subtree) and print the data of each node

void inorder(struct Node\* root) {

if (root != NULL) { // If the root is not NULL

inorder(root->left); // Traverse the left subtree

printf("%d ", root->data); // Print the data of the root

inorder(root->right); // Traverse the right subtree

}

}

// Traverse the binary tree in preorder (root, left subtree, right subtree) and print the data of each node

void preorder(struct Node\* root) {

if (root != NULL) { // If the root is not NULL

printf("%d ", root->data); // Print the data of the root

preorder(root->left); // Traverse the left subtree

preorder(root->right); // Traverse the right subtree

}

}

// Traverse the binary tree in postorder (left subtree, right subtree, root) and print the data of each node

void postorder(struct Node\* root) {

if (root != NULL) { // If the root is not NULL

postorder(root->left); // Traverse the left subtree

postorder(root->right); // Traverse the right subtree

printf("%d ", root->data); // Print the data of the root

}

}

// The main function

int main() {

// Create the binary tree with the root node having data 1

struct Node\* root = createNode(1); // Create the root node with data 1

root->left = createNode(2); // Create the left child of the root node with data 2

root->right = createNode(3); // Create the right child of the root node with data 3

root->left->left = createNode(4); // Create the left child of the left child of the root node with data 4

root->left->right = createNode(5); // Create the right child of the left child of the root node with data 5

// Print the traversals of the binary tree

printf("Inorder traversal: ");

inorder(root);

printf("\n");

printf("Preorder traversal: ");

preorder(root);

printf("\n");

printf("Postorder traversal: ");

postorder(root);

printf("\n");

return 0; // Return 0 to indicate successful execution

}

**OUTPUT**

