**PROGRAM – 7**

***Q. Design and implement binary tree and demonstrate its working.***

* **Theory:**

A binary tree is a tree data structure and is a non-linear data structure in which each parent node can have at most two children. Each node of a binary tree consists of three items:

* Data item
* Address of right child
* Address of left child



**Properties of binary tree:**

1. At each level of i, the maximum number of nodes is 2i.
2. the maximum number of nodes possible at height h is (20 + 21 + 22+….2h) = 2h+1 -1.
3. The minimum number of nodes possible at a height h is equal to h+1.
4. If the number of nodes is minimum, then the height of the tree is maximum and vice-versa.
5. Maximum height is equal to h=n-1.
6. The minimum height is equal to h = log2(n+1) – 1

**Types of binary trees:**

There are five types of Binary tree:

1.Full/ proper/ strict Binary tree

2.Complete Binary tree

3.Perfect Binary tree

4.Degenerate Binary tree

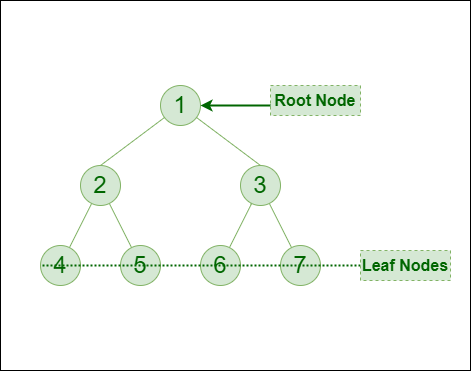
5.Balanced Binarytree

**Tree traversals:**

The process of visiting the nodes is known as tree traversal.

Generally, there are 3 types of tree traversals:

1. Pre-order traversal (current root->left child->right-child)
2. In-order traversal (left child->current root->right child)
3. Post-order Traversal ( left child->right child->current root)



Pre-order traversal: 1-2-4-5-3-6-7

In-order traversal: 4-2-5-1-6-3-7

Post-order traversal: 4-5-2-6-7-3-1

* **Algorithm:**

Step-1: Create a node using structures and initialize members of the structure such as node, left and right.

Step-2: Create a pointer called new node and dynamically allocate memory for it using malloc () function.

Step-3: Point both left and right child address to NULL and add data to data part.

Step-4: Write snippets for inserting roots at the left and right part of the binary tree.

Step-5: Write functions for all the 3 traversals i.e in-order, pre-order and post-order.

Step-6: Insert roots as data given by the user and traverse the tree.

Step-7: END

//----------------------------------------------------

* **Code:**

#include <stdio.h> //header files are included

#include <stdlib.h>

//a structure is created and its members are declared.

struct node {

int item;

struct node\* left;

struct node\* right;

};

// In-order traversal function

void inorderTraversal(struct node\* root) {

if (root == NULL) return; if root is NULL then exit the program

inorderTraversal(root->left); //left root is traversed and printed.

printf("%d ->", root->item); // current root is traversed and printed

inorderTraversal(root->right); //right root is traversed and printed

}

// pre-order Traversal

void preorderTraversal(struct node\* root) {

if (root == NULL) return;

printf("%d ->", root->item); //current root is traversed and printed.

preorderTraversal(root->left); //left root is traversed and printed.

preorderTraversal(root->right); //right root is traversed and printed.

}

// post-order Traversal

void postorderTraversal(struct node\* root) {

if (root == NULL) return;

postorderTraversal(root->left); //left root is traversed and printed.

postorderTraversal(root->right); //right root is traversed and printed.

printf("%d ->", root->item); //current root is traversed and printed.

}

// Create a new Node

struct node\* createNode(value) {

//dynamically memory is allocated

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

newNode->item = value; //item is inserted

newNode->left = NULL; // Left link is pointed to NULL

newNode->right = NULL; // Left link is pointed to NULL

return newNode; //node is returned

}

// Insert on the left of the node

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

root->left = createNode(value); //root is inserted at left

return root->left;

}

// Insert on the right of the node

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

root->right = createNode(value); //root is inserted at left

return root->right;

}

int main() {

struct node\* root = createNode(1);

insertLeft(root, 12);

insertRight(root, 9);

insertLeft(root->left, 5); //values or roots given by user.

insertRight(root->left, 6);

printf("Pre-order Traversal is \n"); //traversal is printed

preordertraversal(root);

printf("\n");

printf("\nIn-order Traversal is \n"); //traversal is printed

inordertraversal(root);

printf("\n");

printf("\nPost-order Traversal is \n"); //traversal is printed

postordertraversal(root);

printf("\n");

}

//END OF THE PROGRAM

//-------------------------------------------------

**“SCREENSHOTS OF THE OUTPUT”**

