Experiment -11

Operations on Binary tree.

→

```
#include <stdio.h>
#include <stdlib.h>
 struct node {
    int data;
    struct node *left;
    struct node * right;
};

Struct node * create (int val);
struct node * insert left (struct node * root, int
struct node * insert right (struct node * root, int val
void delete subtree (struct node * root);
void delete left (struct node * root);
void delete right (Struct node * root);
void display (struct node * root);

int main() {
   Struct node * root = create (1);
    insert left (root, 2);
    insert right (root, 3);
    insert left (root → left, 4);
    insert right (root → left, 5);
    insert left (root → right, 6);
    insert right (root → right, 7);
    insert left (root → left → left, 8);

    printf ("Original tree: ");
     display (root);
     printf ("\n");
```

```
    delete left(root)
    printf("Tree after deleting left subtree of
            root:");
    display(root);
    return 0;
}

struct node *create(int val){
    struct node *new = (struct node*)malloc
            (sizeof(struct node));
    if(new==NULL){
        printf("Memory alloc failed");
        exit(1);
    }

    new->data = val;
    new->left = NULL;
    new->right = NULL;
    return new;
}

struct node *insert right(struct node*root, in
    val);
    if(root == NULL){
    printf("Can't insert in NULLroot");
     return NULL;
}
root -> right = create(val);
 return root->right;
}
```

```
Struct node * insert left (struct node* root,
    int val);
{
    if(root == NULL){
        printf("Can't insert in NULL");
        return NULL;
    }
    root->left = create(val);
    return root -> left;
}
void delete subtree (struct node * node){
    if(node == NULL){
        return 0;
    }
    delete subtree(node->left);
    delete subtree (node->right);
    free (node);
}
void delete left (struct node *root){
    if(root == NULL || root->left == NULL){
        printf("Nothing to delete");
        return;
    }
    delete subtree (root->left);
    root->left = NULL;
    printf("In left subtree deleted");
}
void deleteright (struct node * root) {
    if(root == NULL || root->right == NULL){
        printf("Nothing to delete");
        return 0;
```

```
    delete subtree (root->right);
    root->right = NULL;
}
void display (struct node *root){
    if (root == NULL){
        return;
    }
    printf("%d", root->data);
    display(root->left);
    display(root->right);
}
```

→ O/P

Original tree = 1 2 4 8 5 3 6 7
left subtree deleted.
Tree after deletion = 1 3 6 7

(2) ~~Pre order, Post order, In order, Traversal.~~

→ ~~#include <stdio.h>~~
~~#include <stdlib.h>~~
~~main()~~