//Implementation of binary tree with different operations

```
#include<stdio.h>
#include<stdlib.h>

struct node
{

        int data;
        struct node *left,*right;
};
struct node *root=NULL;
int level=-1;

void create()
{
                if(root==NULL)
                {
                struct node *temp = (struct node*)malloc(sizeof(struct node));
                int value;
                printf("Enter a value : ");
                scanf("%d",&value);
                temp->data = value;
                temp->left = NULL;
                temp->right = NULL;
                root = temp;
                level = 0;
                }
                else
                {
                printf("Root already exists");
                }
}

void Insert()
{
        if(root==NULL){
                printf("Root is NULL");
                printf("Create the tree to insert elements.");
                create();
        }
        else{
                struct node *temp = (struct node*)malloc(sizeof(struct node));
                int value;
                printf("Enter any value : ");
                scanf("%d",&value);
                temp->data = value;
                temp->left = NULL;
                temp->right = NULL;

                if(root->left == NULL || root->right == NULL)
                {
                        if(root->left == NULL){
                                root->left = temp;
```

```
                        }
                        else if(root->right == NULL){
                                root->right = temp;
                        }

                        level = 1;
                }

                else if(level ==1 || level == 2)
                {
                        if((root->left)->left == NULL){
                                (root->left)->left = temp;
                        }

                        else if((root->left)->right == NULL){
                                (root->left)->right = temp;
                        }

                        else if((root->right)->left == NULL){
                                (root->right)->left = temp;
                        }

                        else if((root->right)->right == NULL){
                                (root->right)->right = temp;
                        }

                        level = 2;
                }
        }
}
void preorder(struct node *temp)
{
        if(temp!=NULL)
        {
                printf("%d ",temp->data);

                if(temp->left)
                        preorder(temp->left);

                if(temp->right)
                        preorder(temp->right);
        }
        else{
                printf("tree doesnot exist");
                return;
        }
}

void inorder(struct node *temp)
{
        if(temp!=NULL)
        {
                if(temp->left)
                        inorder(temp->left);
```

```
                printf("%d ",temp->data);

                if(temp->right)
                        inorder(temp->right);
        }
        else{
                printf("tree doesnot exist");
                return;
        }
}

void postorder(struct node *temp)
{
        if(temp!=NULL)
        {
                if(temp->left)
                        postorder(temp->left);

                if(temp->right)
                        postorder(temp->right);

                printf("%d ",temp->data);
        }
        else{
                printf("tree doesnot exist");
                return;
        }
}
void deleteTree(struct node* temp)
{
   if (temp == NULL) return;

   /* first delete both subtrees */
   deleteTree(temp->left);
   deleteTree(temp->right);

   /* then delete the node */
   printf("\n Deleting node: %d", temp->data);
   free(temp);
}
void mirror(struct node* node)
{
 if (node==NULL)
   return;
 else
 {
   struct node* temp;

   /* do the subtrees */
   mirror(node->left);
   mirror(node->right);

   /* swap the pointers in this node */
   temp       = node->left;
   node->left  = node->right;
```

```
    node->right = temp;
  }
}
void printLevelOrder(struct node* root)
{
    int h = height(root);
    int i;
    for (i = 0; i <= h; i++)
        printCurrentLevel(root, i);
}
void printCurrentLevel(struct node* root, int level)
{
    if (root == NULL)
        return;
    if (level == 1)
        printf("%d ", root->data);
    else if (level > 1) {
        printCurrentLevel(root->left, level-1);
        printCurrentLevel(root->right, level-1);
    }
}
int height(struct node* node)
{
    if (node == NULL)
        return 0;
    else {
        /* compute the height of each subtree */
        int lheight = height(node->left);
        int rheight = height(node->right);

        /* use the larger one */
        if (lheight > rheight)
            return (lheight +1);
        else
            return (rheight +1);
    }
}

int main()
{
            int ch,dis;
            while(1)
            {

            printf("\n1.Create\n2.Insert\n3.Display\n4.DeleteTree\n5.mirror\n6.levelorder\n0.EXIT\n");
                        printf("Enter your choice : ");
                        scanf("%d",&ch);
                        switch(ch)
                        {
                                    case 1: create(); break;
                                    case 2: Insert(); break;
                                    case 3: printf("1.Preorder\n2.Inorder\n3.Postorder\n");
                                                            printf("Enter your choice : ");
                                                            scanf("%d",&dis);
                                                            switch(dis)
```

```
                                                {
                                                        case 1: preorder(root); break;
                                                        case 2: inorder(root); break;
                                                        case 3: postorder(root); break;
                                                        default : printf("Choose the correct option.");
break;
                                                }
                                                break;
                                case 4: deleteTree(root);
                                        root=NULL;
                                        break;
                                case 5: mirror(root);
                                        break;
                                case 6: printLevelOrder(root);
                                        break;
                                case 0: return 0;
                                default : printf("Choose the correct option."); break;
                        }
                }
        }
```