- Tree Implementation Using linked list.

```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
#include<process.h>
struct node
{
node *left;
int info;
node *right;
}*root;
node * create();//1
void insert(int);//2
void preorder(node *);//3
void inorder(node *);//4
void postorder(node *);//5
void check_node(node *, int);//6
int  count_leaf(node *);//7
int  count_total(node *);//8
void search(int);//9
int find_max(node*);//10
int find_min(node*);//11
void dis_even(node*);//12
void dis_odd(node*);//13
int find_level(node*, int, int);//14
void del_leaf(node*,node*);//15
void del_one(node*,node*);//16
void del_two(node*);//17
void del(int);//18void main()
```

```
{
    clrscr();
    int ch,x,srch,a,z;
    node *p;
    do
    {
            printf("\nEnter Your Choise:");
            printf("\n1)Insert\n2)Preorder\n3)Inorder\n4)Postorder\n5)Check_Node(Leaf,Having one child,Having two child)\n6)Count_Leaf\n7)Count_total\n8)Search Node\n9)Find Max_Node\n10)Find Min_Node\n11)Display Even Nodes\n12)Display Odd Nodes\n13)Find level_Node\n14)Delete\n15)Exit\n");
            scanf("%d",&ch);
            switch(ch)
            {
                    case 1:printf("Enter Your Data:");
                            scanf("%d",&x);
                            insert(x);
                            printf("%d is Inserted..\n",x);
                            break;

                    case 2:preorder(root);
                            break;

                    case 3:inorder(root);
                            break;

                    case 4:postorder(root);
                            break;

                    case 5:printf("\nEnter Node Value to Check:");
```

```
        scanf("%d", &x);
        check_node(root,x);
        break;

case 6:z=count_leaf(root);
        printf("\nTotal Leaf Nodes=%d",z);
        break;

case 7:z=count_total(root);
        printf("\nTotal Node Count=%d",z);
        break;

case 8:printf("\nEnter Value to Search:");
        scanf("%d",&srch);
        search(srch);
        break;

case 9:z=find_max(root);
        printf("\nMax Value:%d",z);
        break;

case 10:z=find_min(root);
            printf("Min Value:%d",z);
            break;

case 11:printf("\nEven Values:");
            dis_even(root);
            break;
```

```
                        case 12:printf("\nOdd Values:");
                                dis_odd(root);
                                break;

                        case 13:printf("\nEnter Search Val:\n");
                                scanf("%d", &a);
                                printf("\nLevel of %d = %d\n", a, find_level(root, a, 1));
                                break;
                        case 14:printf("Enter val to delete=");
                                scanf("%d",&a);
                                del(a);
                                break;
                        case 15:exit(1);
                    }
                  }while(ch!=15);
getch();
}
node * create()
{
        node *p;
        p=(node *)malloc(sizeof(node));
        return(p);
}

void insert(int x)
{
        node *p,*temp;
        p=create();
        p->left=NULL;
```

```
p->info=x;
p->right=NULL;
if(root==NULL)
{
        root=p;
        printf("\nNode is Inserted.\n");
}
else
{
        temp=root;
        while(temp != NULL)
        {
                if(p->info < temp->info)
                {
                        if(temp->left == NULL)
                        {
                                temp->left=p;
                                printf("\nNode is Inserted.\n");
                                break;
                        }
                        else
                        {
                                temp=temp->left;
                        }
                }
                else if(p->info > temp->info)
                {
                        if(temp->right == NULL)
                        {
```

```
                                        temp->right=p;
                                        printf("\nNode is Inserted.\n");
                                        break;
                                }
                                else
                                {
                                        temp=temp->right;
                                }
                        }
                }
        }
}
void preorder(node *p)//VLR
{
        if(p!=NULL)
        {
                printf("\t%d",p->info);
                preorder(p->left);
                preorder(p->right);
        }
}
void inorder(node *p) //LVR
{
        if(p!=NULL)
        {
                inorder(p->left);
                printf("\t%d",p->info);
                inorder(p->right);
        }
```

```
}
void postorder(node *p)//LRV
{
	if(p!=NULL)
	{
		postorder(p->left);
		postorder(p->right);
		printf("\t%d",p->info);
	}
}
int count_leaf(node *p)
{
	if(p==NULL)
	{
		return(0);
	}
	else if(p->left == NULL && p->right == NULL)
	{
		printf("\t%d",p->info);
		return(1);
	}
	else
	{
		return(count_leaf(p->left) + count_leaf(p->right));
	}
}
int count_total(node *p)
{
	if(p==NULL)
```

```
        {
                return(0);
        }
        else if(p->left == NULL && p->right == NULL)
        {
                return(1);
        }
        else
        {
                return(count_total(p->left) + count_total(p->right) + 1);
        }
}
void search(int srch)
{
        int f = 0;
        node *p;
        p=root;
        while(p!=NULL)
        {

                if(srch == p->info)
                {
                        f=1;
                        break;
                }
                else if(srch < p->info)
                {
                        p=p->left;
                }
```

```
                else if(srch > p->info)
                {
                        p=p->right;
                }

        }
        if(f == 1)
        {
                printf("\nNode is Found.");
        }
        else
        {
                printf("\nNode is not Found.");
        }
}
void del(int   x)
{
   int   f=0;
   node   *c,*p;
   p=c=root;
   while(c!=NULL)
   {
        if(x==c->info)
        {
           f=1;
                break;
        }
        else if(x<c->info)
        {
```

```
            p=c;
            c=c->left;
        }
        else   if(x>c->info)
        {
            p=c;
            c=c->right;
        }
    }
    if(f==1)
    {
        if(c->left==NULL&&c->right==NULL)
        {
            del_leaf(p,c);
        }
        else  if(c->left!=NULL&&c->right!=NULL)
        {
            del_two(c);
        }
        else
        {
            del_one(p,c);
        }

    }
    else
    {
        printf("\nNode is NOT found ");
    }
```

```
}
void del_two(node   *c)
{
   node  *lft,*p=NULL;
   lft=c->left;
   while(lft->right!=NULL)
   {
            p=lft;
         lft=lft->right;
   }
   c->info=lft->info;
   if(p==NULL)
   {
    p=c;
   }
   if(lft->left==NULL&&lft->right==NULL)
   {
         del_leaf(p,lft);
   }
   else
   {
         del_one(p,lft);
   }
}
void del_one(node   *p,node  *c)
{
   if(c==p->left)
   {
         if(c->left!=NULL)
```

```
            {
               p->left=c->left;
            }
            else
            {
               p->left=c->right;
            }
    }
    else  if(c==p->right)
    {
            if(c->right!=NULL)
            {
               p->right=c->right;
            }
            else
            {
               p->right=c->left;
            }
    }
    free(c);
    printf("\nNode is deleted..");
}
void del_leaf(node  *p,node *c)
{
    if(c==p->left)
    {
            p->left=NULL;
    }
    else  if(c==p->right)
```

```
    {
        p->right=NULL;
    }
    free(c);
    printf("\n Node is deleted...");
}
int find_level(node *p, int key, int level)
{
    if(p == NULL)
    {
        return 0;
    }
    else if(p->info == key)
    {
        return level;
    }
    else if(key < p->info)
    {
        return find_level(p->left, key, level + 1);
    }
    else
    {
        return find_level(p->right, key, level + 1);
    }
}
void dis_odd(node *p)
{
    if(p != NULL)
    {
```

```
        dis_odd(p->left);
        if(p->info % 2 != 0)
        {
           printf("\t%d", p->info);
        }
        dis_odd(p->right);
   }
}
void dis_even(node *p)
{
   if(p != NULL)
   {
        dis_even(p->left);
        if(p->info % 2 == 0)
        {
           printf("\t%d", p->info);
        }
        dis_even(p->right);
   }
}
int find_min(node *p)
{
   if(p == NULL)
   {
        return 0;
   }
   else
   {
        while(p->left != NULL)
```

```
        {
            p = p->left;
        }
        return p->info;
    }
}
int find_max(node *p)
{
    if(p == NULL)
    {
        return 0;
    }
    else
    {
        while(p->right != NULL)
        {
            p = p->right;
        }
        return p->info;
    }
}
void check_node(node *temp, int key)
{
    temp = root;
    while(temp != NULL)
    {
        if(key == temp->info)
        {
            if(temp->left == NULL && temp->right == NULL)
```

```
            {
                printf("\nThe node is a leaf node\n", key);
            }
            else if(temp->left == NULL || temp->right == NULL)
            {
                printf("\nThe node has one child.\n", key);
            }
            else
            {
                printf("\nThe node has two children.\n", key);
            }
            return;
        }
        else if(key < temp->info)
        {
            temp = temp->left;
        }
        else
        {
            temp = temp->right;
        }
    }
    printf("\nNode is not found\n", key);
}
```