/*CREATION OF A BINARY SEARCH TREE USING RECURSION AND IT'S TRAVERSAL*/

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

typedef struct btree
{
int item;
struct btree *lchild,*rchild;
}node;
node *create(node*,int);
void inorder(node*);
void preorder(node*);
void postorder(node*);
int countleave(node*);
int countnode(node*);
void depth(node*);
int total1,total2;
void main()
{
int n,x,count,d;
node *root=NULL;
clrscr();
while(1)
{
printf("\n 1:creation\n 2:inorder\n 3:preorder\n 4:postorder"
     "\n 5:countleave\n 6:countnode\n 7:depth\n 8:exit\n");
scanf("%d",&n);
switch(n)
{
case 1: printf("Enter the element\n");
           scanf("%d",&x);
           root=create(root,x);
           break;
case 2: inorder(root);
           break;
case 3: preorder(root);
           break;
case 4: postorder(root);
           break;
case 5: count=countleave(root);
           printf("The no of leave nodes is:%d",count);
           break;
case 6: count=countnode(root);
           printf("The no of nodes is:%d",count);
           break;
case 7: depth(root);
```

```
            if(total1>total2)
            d=total1+1;
            else
            d=total2+1;
            printf("Depth of binary search tree is:%d",d);
            break;
case 8: exit(0);
default:printf("Sorry wrong selection try once again\n");
}
}
}
node* create(node *t,int x)
{
if(t==NULL)
{
node *cur;
cur=(node*)malloc(sizeof(node));
cur->item=x;
cur->lchild=NULL;
cur->rchild=NULL;
return cur;
}
else
if(t->item<x)
t->rchild=create(t->rchild,x);
else
t->lchild=create(t->lchild,x);
return t;
}

void inorder(node *temp)
{
if(temp!=NULL)
{
inorder(temp->lchild);
printf("%3d",temp->item);
inorder(temp->rchild);
}
}
void preorder(node *temp)
{
if(temp!=NULL)
{
printf("%3d",temp->item);
preorder(temp->lchild);
preorder(temp->rchild);
}
}
```

```
void postorder(node *temp)
{
if(temp!=NULL)
{
postorder(temp->lchild);
postorder(temp->rchild);
printf("%3d",temp->item);
}
}

int countleave(node *t)
{
 static int c;
 if(t!=NULL)
 {
 if(t->lchild==NULL&&t->rchild==NULL)
 c++;
 countleave(t->lchild);
 countleave(t->rchild);
 }
 return c;
 }
int countnode(node *t)
{
 static int c;
 if(t!=NULL)
  {
  c++;
 if(t->lchild!=NULL)
   countnode(t->lchild);
 if(t->rchild!=NULL)
  countnode(t->rchild);
 }
 return c;
}
void depth(node *t)
{
 if(t!=NULL)
  {
  if(t->lchild!=NULL)
  {
  total1++;
  depth(t->lchild);
  }
  if(t->rchild!=NULL)
  {
  total2++;
  depth(t->rchild);
```

```
  }
  }
}
```