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

typedef struct BSTnode
{
 int data;
 struct BSTnode* left;
 struct BSTnode* right;
}BSTnode;

BSTnode* insert(BSTnode* T,int x)
{
 BSTnode* temp;
 if(T==NULL)
   {
    //create node
    temp = (BSTnode*)malloc(sizeof(BSTnode));
    temp->data = x;
    temp->left = NULL;
    temp->right = NULL;
    return temp;
   }
 else if(x<T->data)
   {
    //go to left subtree
    T->left = insert(T->left,x);
   }
   else if(x>=T->data)
   {
    //go to right subtree
    T->right = insert(T->right,x);
   }
   return T;
}

void inorder(BSTnode* T)
{
 if(T!=NULL)
   {
    inorder(T->left);
    printf("%d ",T->data);
    inorder(T->right);
   }
}

void preorder(BSTnode* T)
{
 if(T!=NULL)
   {
    printf("%d ",T->data);
    preorder(T->left);
    preorder(T->right);
   }
```

```
}

void postorder(BSTnode* T)
{
 if(T!=NULL)
   {
    postorder(T->left);
    postorder(T->right);
    printf("%d ",T->data);
   }
}

void print(BSTnode* T)
{
 if(T!=NULL)
   {
    printf("%d ",T->data);
    print(T->left);
    print(T->right);
   }
}

BSTnode* convertmirror(BSTnode* T)
{
 BSTnode* temp;
 if(T!=NULL)
   {
    temp = T->left;
    T->left = convertmirror(T->right);
    T->right = convertmirror(temp);
   }
 return T;
}

int total(BSTnode* T)
{
 int sum;
 if(T == NULL)
   sum = 0;
 else
   sum = T->data + total(T->left) + total(T->right);

 return sum;
}


int oddtotal(BSTnode* T)
{
 int sum;

 if(T==NULL)
   sum = 0;
 else if(T->data%2 !=0)
   sum = T->data + oddtotal(T->left) + oddtotal(T->right);
```

```
 else
   sum = oddtotal(T->left) + oddtotal(T->right);

 return sum;
}

int eventotal(BSTnode* T)
{
 int sum;

 if(T==NULL)
   sum = 0;
 else if(T->data%2 ==0)
   sum = T->data + eventotal(T->left) + eventotal(T->right);
 else
   sum = eventotal(T->left) + eventotal(T->right);


 return sum;
}



int count(BSTnode* T)
{
 int cnt;
 if(T==NULL)
   cnt = 0;
 else
   cnt = 1 + count(T->left) + count(T->right);

 return cnt;
}

int countleaf(BSTnode* T)
{
 int cnt;
 if(T==NULL)
   cnt = 0;
 else if(T->left==NULL && T->right==NULL)
   {
    printf("%d ",T->data);
    cnt = 1 + countleaf(T->left) + countleaf(T->right);
   }
 else
   cnt = countleaf(T->left) + countleaf(T->right);

 return cnt;
}


BSTnode* copy(BSTnode* T)
{
 BSTnode* T1=NULL;
```

```
 if(T!=NULL)
   {
    T1 = (BSTnode*)malloc(sizeof(BSTnode));
    T1->data = T->data;
    T1->left = copy(T->left);
    T1->right = copy(T->right);
   }
 return T1;
}

int compare(BSTnode* T1,BSTnode* T2)
{
 if(T1==NULL && T2==NULL)
   return 1;
 if(T1!=NULL && T2!=NULL)
   {
    if(T1->data==T2->data && compare(T1->left,T2->left) && compare(T1->right,T2->right))
    return 1;
   }
 return 0;
}

void degree(BSTnode* T)
{
 if(T!=NULL)
   {
    if(T->left!=NULL && T->right!=NULL)
      printf("\nDegree of %d is 2",T->data);
    else if((T->left==NULL && T->right!=NULL) || (T->left!=NULL && T->right==NULL))
      printf("\nDegree of %d is 1",T->data);
    else
      printf("\nDegree of %d is 0",T->data);
   degree(T->left);
   degree(T->right);
   }
}


void search(BSTnode* T,int x)
{
 int flag = 0;
 while(T!=NULL)
      {
       if(x==T->data)
     {
      flag = 1;
      break;
     }
       else if(x<T->data)
     T = T->left;
       else if(x>=T->data)
     T = T->right;
      }
 if(flag==0)
```

```
   printf("%d not found",x);
 else
   printf("%d found",x);
}

BSTnode *delet(BSTnode *T,int x)
{
 BSTnode *temp;
 if(T==NULL)
   {
    printf("\nElement not found :");
    return(T);
   }
 if(x < T->data)                // delete in left subtree
   {
    T->left=delet(T->left,x);
    return(T);
   }
 if(x > T->data)                // delete in right subtree
   {
    T->right=delet(T->right,x);
    return(T);
   }

 // element is found
 if(T->left==NULL && T->right==NULL)   // a leaf node
   {
    temp=T;
    free(temp);
    return(NULL);
   }
 if(T->left==NULL)
   {
    temp=T;
    T=T->right;
    free(temp);
    return(T);
   }
 if(T->right==NULL)
   {
    temp=T;
    T=T->left;
    free(temp);
    return(T);
   }
 // node with two children
 //go to the inorder successor of the node
 temp=T->right;
 while(temp->left !=NULL)
       temp=temp->left;
 T->data=temp->data;
 T->right=delet(T->right,temp->data);
 return(T);
}
```

```
int heightoftree(BSTnode* T)
{

  int h=-1;
  int leftheight,rightheight;

  if(T!=NULL)
  {
    leftheight = heightoftree(T->left);
    rightheight = heightoftree(T->right);
    if(leftheight > rightheight)
      h = leftheight + 1;
    else
      h = rightheight + 1;
  }
  return h;
}

void printlevel(BSTnode* T,int level)
{
  if(T!=NULL)
    {
      if(level==0)
    printf("%d",T->data);
      else if(level>0)
    {
      printlevel(T->left,level-1);
      printlevel(T->right,level-1);
    }
    }
}

int main()
{
 int n,i,x,ch,sum,ans,cnt;
 BSTnode* root=NULL,*root1=NULL;
 clrscr();
 printf("\nEnter how many nodes:");
 scanf("%d",&n);
 for(i=1;i<=n;i++)
    {
     printf("\nEnter data for node:");
     scanf("%d",&x);
     root = insert(root,x);
    }
 do
 {
 printf("\n1:Inorder");
 printf("\n2:Preorder");
 printf("\n3:Postorder");

 printf("\n4:Print");
```

```
printf("\n5:MirrorImage");

printf("\n6:Count nodes");
printf("\n7:Insert");
printf("\n8:Delete");

printf("\n9:Count Leaf node");
printf("\n10:Degree of each node");
printf("\n11:Search a node");

printf("\n12:Sum of all numbers from BST");
printf("\n13:Sum of all even numbers from BST");
printf("\n14:Sum of all odd numbers from BST");

printf("\n15:Copy BST into another BST");
printf("\n16:Compare two BST");

printf("\n17: Height of BST");
printf("\n18: Level wise Display");

printf("\n19:Quit");
printf("\nEnter your choice:");
scanf("%d",&ch);
switch(ch)
      {
   case 1: inorder(root);
       break;
   case 2: preorder(root);
       break;
   case 3: postorder(root);
       break;
   case 4: print(root);
       break;
   case 5: root = convertmirror(root);
       break;
   case 6: cnt = count(root);
       printf("\nNumber of nodes is %d",cnt);
       break;
   case 7: printf("\nEnter data for node");
       scanf("%d",&x);
       root = insert(root,x);
       break;
   case 8: printf("\nEnter node to be deleted:");
       scanf("%d",&x);
       root = delet(root,x);
       break;
   case 9: cnt = countleaf(root);
       printf("\nNumber of leaf nodes are %d",cnt);
       break;
   case 10:degree(root);
       break;
   case 11:printf("\nEnter data to be searched:");
       scanf("%d",&x);
       search(root,x);
```

```
        break;
    case 12:sum = total(root);
        printf("\nSum of all numbers is %d",sum);
        break;
    case 13:sum = eventotal(root);
        printf("\nSum of even numbers is %d",sum);
        break;
    case 14:sum = oddtotal(root);
        printf("\nSum of odd numbers is %d",sum);
        break;
    case 15:root1 = copy(root);
        printf("\nCopied succesfully");
        printf("\nOld BST\n");
        print(root);
        printf("\nCopid BST\n");
        print(root1);
        break;
    case 16:ans = compare(root,root1);
        if(ans==1)
          printf("\nBoth BST are equal");
        else
          printf("\nBoth BST are not equal");
        break;
    case 17: ans = heightoftree(root);
         if(ans!=-1)
            printf("\nHeight of Tree is %d",ans);
         else
            printf("\nTree is empty");
         break;
    case 18: ans = heightoftree(root);
         for(i=0;i<=ans;i++)
            {
            printlevel(root,i);
            printf("\n");
            }

         break;
       }
 }while(ch!=19);

 return 0;
}
```