Name:- Amit Bandu Swami

Roll No. :- 2221018

Class :- SE COMP

Ass 1

Problem :- Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree - i. Insert new node, ii. Find number of nodes in longest path from root, iii. Minimum data value found in the tree, iv. Change a tree so that the roles of the left and right pointers are swapped at every node, v. Search a value

```
#include<iostream>
using namespace std;
class node
{
   public:
     int data;
     node *l,*r;
};
class bst
{
   char ans;
   public:
     void create(node *root);
     void inorder(node *root);
     void preorder(node *root);
     void postorder(node *root);
     void insert(node *root);
     void min(node *root);
     void max(node *root);
     int height(node *root);
     void mirror(node *root);
     void search(node *root);
     int count(node *root);
     node *del(node *root,int key);
     node *smallest(node *root);
};
void bst::create(node *root)
{
   node *ne,*temp;
   do
   {
      ne=new node;
```

```
cout<<"Enter new node data:"<<endl;
```

```
      cin>>ne->data;
      ne->l=NULL;
      ne->r=NULL;
      temp=root;
      while(1)
      {
         if(ne->data<temp->data)
         {
           if(temp->l==NULL)
           {
             temp->l=ne;
             break;
           }
           else
           temp=temp->l;
         }
         else
         {
            if(temp->r==NULL)
           {
             temp->r=ne;
             break;
           }
           else
           temp=temp->r;
         }
      }
      cout<<"Do you want to add another node:(y/n)"<<endl;
      cin>>ans;
   }while(ans=='y');
}
void bst::inorder(node *root)
{
   if(root!=NULL)
   {
      inorder(root->l);
      cout<<root->data<<" ";
      inorder(root->r);
   }
}
void bst::preorder(node *root)
{
   if(root!=NULL)
   {
      cout<<root->data<<"  ";
      preorder(root->l);
      preorder(root->r);
   }
}
void bst::postorder(node *root)
{
   if(root!=NULL)
   {
      postorder(root->l);
      postorder(root->r);
      cout<<root->data<<"  ";
   }
}
void bst::insert(node *root)
{
   node *ne,*temp;
   ne=new node;
   cout<<"Enter new node data:"<<endl;
   cin>>ne->data;
   ne->l=NULL;
   ne->r=NULL;
   temp=root;
   while(1)
   {
      if(ne->data<temp->data)
      {
         if(temp->l==NULL)
         {
            temp->l=ne;
            break;
         }
         else
         temp=temp->l;
      }
      else
      {
         if(temp->r==NULL)
         {
            temp->r=ne;
            break;
         }
         else
         temp=temp->r;
      }
   }
```

```
}

void bst::min(node *root)
{
   node *temp;
   temp=root;
   while(temp->l!=NULL)
   {
     temp=temp->l;
   }
   cout<<"Minimum is: "<<temp->data;
}
void bst::max(node *root)
{
   node *temp;
   temp=root;
   while(temp->r!=NULL)
   {
     temp=temp->r;
   }
   cout<<"Maximum is: "<<temp->data;
}
int bst::height(node *root)
{
   int i=1,j=1,max=0;
   if(root!=NULL)
   {
      i=i+height(root->l);
      j=j+height(root->r);

   if(i>j)
   {
      max=i;
   }
   else
   max=j;
   }
   return max;
}
void bst::mirror(node *root)
{
   node *temp;
   if(root!=NULL)
   {
      temp=root->l;
```

```
    root->l=root->r;
    root->r=temp;
    mirror(root->l);
    mirror(root->r);
  }
}
void bst::search(node *root)
{
  node *temp;
  int key,flag=0;
  cout<<"Enter data to search:"<<endl;
  cin>>key;
  temp=root;
  while(temp!=NULL)
  {
    if(key==temp->data)
    {
      cout<<key<<" is present!!"<<endl;
      flag=1;
      break;
    }
    else if(key<temp->data)
    {
      temp=temp->l;
    }
    else
    {
      temp=temp->r;
    }
  }
  if(flag==0)
  {
    cout<<key<<" is absent!!"<<endl;
  }
}
int bst::count(node *root)
{
 int i=1,j=1;
 if(root==NULL)
 return 0;

 if(root!=NULL)
 {
  i=count(root->l);
  j=count(root->r);
  }
  return 1+i+j;
}
node* bst::smallest(node *root)
{
   node *temp;
   temp=root;
   while(temp->l!=NULL)
   {
      temp=temp->l;
   }
   return temp;
}
node* bst::del(node *root,int key)
{
   node *small;
   if(root==NULL)
   return root;

   if(key<root->data)
   root->l=del(root->l,key);
   else if(key>root->data)
   root->r=del(root->r,key);
   else
   {
      if(root->r!=NULL)
      {
         small=smallest(root->r);
         root->data=small->data;
         root->r=del(root->r,small->data);
      }
      else
      {
         return root->l;
      }
   }
   return root->l;
}
int main()
{
   bst ob;
   node *root,*d;
   int ch,h,c,key;
   while(1)
   {
```

```
cout<<"1. Create" <<endl;
cout<<"2. Inorder"<<endl;
cout<<"3. Preorder"<<endl;
cout<<"4. Postorder"<<endl;
cout<<"5.  Insert"<<endl;
cout<<"6. Minimum"<<endl;
cout<<"7. maximum"<<endl;
cout<<"8. Height"<<endl;
cout<<"9. Mirror"<<endl;
cout<<"10. Search"<<endl;
cout<<"11. Count"<<endl;
cout<<"12. Delete"<<endl;
cout<<"Enter your choice:"<<endl;
cin>>ch;
switch(ch)
{
   case 1:root=new node;
        cout<<"Enter the root data:"<<endl;
        cin>>root->data;
        root->l=NULL;
        root->r=NULL;
        ob.create(root);
        break;
   case 2:ob.inorder(root);
        break;
   case 3:ob.preorder(root);
        break;
   case 4:ob.postorder(root);
        break;
   case 5:ob.insert(root);
        break;
   case 6:ob.min(root);
        break;
   case 7:ob.max(root);
        break;
   case 8:h=ob.height(root);
        cout<<"Height of a tree is: "<<h<<endl;
        break;
   case 9:ob.mirror(root);
        ob.inorder(root);
        break;
   case 10:ob.search(root);
        break;
   case 11:c=ob.count(root);
         cout<<"Total no. of nodes are: "<<c<<endl;
```

```
                break;
          case 12:cout<<"Enter key to delete:"<<endl;
                cin>>key;
                d=ob.del(root,key);
                break;
        }
    }
}
```