Assignment no.4

```
/*Title : 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
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 */
```

INPUT:

```
#include<iostream>
#include<stdlib.h>
using namespace std;

class bstnode
{
    public:

    int data;
    bstnode *left,*right;
    bstnode(int x)
    {
        data=x;
        left=right=NULL;
    }
};

class bst
{
    bstnode*root;
    public:
    bst()
    {
        root=NULL;
    }

        bstnode*create();
        void insert(int x);
        bstnode*find(int x);
        bstnode*minvalue(bstnode*root);
        bstnode*maxvalue(bstnode*root);
        int longest_path(bstnode*T);
        void display(bstnode*t);
        bstnode*mirror(bstnode*t);
};

bstnode*bst::create()
{
    int x,i,n;
    root=NULL;
    cout<<"enter total number of nodes:";
    cin>>n;
    cout<<"enter tree value:";
    for(i=0;i<n;i++)
    {
        cin>>x;
        insert(x);
    }
    return(root);
}

void bst::insert(int x)

{
    bstnode *p,*q,*r;
    r=new bstnode (x);
    if(root==NULL)
    {
        root=r;
        return;
    }
    p=root;                    // root!=null
    while(p!=NULL)
    {
        q=p;
        if(x>p->data)
            p=p->right;
        else
            p=p->left;
    }
    if(x>q->data)
        q->right=r;
    else
        q->left=r;
}
```

```
bstnode*bst::find(int x)
{
    while(root!=NULL)
    {
        if(x==root->data)
        return (root);
        if(x>root->data)
        root=root->right;
        else
        root=root->left;
    }
    return NULL;
}


bstnode *bst:: minvalue(bstnode*root)
            {
                while(root->left!=NULL)
                {
                    root=root->left;

                }
                cout<<root->data;
            }

bstnode *bst:: maxvalue(bstnode*root)
                    {
                        while(root->right!=NULL)
                        {
                            root=root->right;

                        }
                        cout<<root->data;
                    }


int bst::longest_path(bstnode*T)
{
    int hl,hr;
    if(T==NULL)
    return(0);
    if(T->left==NULL && T->right==NULL)
    return(0);
    hl=longest_path(T->left);
    hr=longest_path(T->right);
    if(hl>hr)
    {
        return(hl+1);
    }
    else
    {
        return(hr+1);
    }
}


void bst::display(bstnode *t)
{
    if(t!=NULL)
    {
        display(t->left);
        cout<<"\t"<<t->data;
        display(t->right);
    }
}

bstnode*mirror(bstnode*t)
{
    bstnode*temp;
    if(t!=NULL)
    {
        temp=t->left;
        t->left=mirror(t->right);
        t->right=mirror(temp);

    }
    return(t);
}

int main()
{
    int ch,x,i;
    bst b;
    bstnode*p,*q,*root;

    do
    {
        cout<<"\n1.create \n2.find \n3.find_min 
\n4.find_max\n5.longest_path\n6.display\n7.mirror";
        cout<<"\nenter u r choice : ";
        cin>>ch;
        switch(ch)
        {
            case 1:
            root=b.create();
            break;

            case 2:
            cout<<"enter node to be searched ";
            cin>>x;
            p=b.find(x);
            if(p==NULL)
            cout<<"\nnode not found ";
            else
            cout<<"node found"<<p->data;
            break;

            case 3:

            cout<<"\n The minimum value = ";
            b.minvalue(root);
            break;

            case 4:

            cout<<"\n The maximum value = ";
            b.maxvalue(root);
            break;

            case 5:
            i=b.longest_path(root);
            cout<<" longest path in tree  "<<i+1;
            break;


            case 6:
```

```
            b.display(root);
            break;

            case 7:
            mirror(root);
            break;


        }
    }
    while(ch!=8);
    return 0;
}
```

OUTPUT:

```
1.create
2.find
3.find_min
4.find_max
5.longest_path
6.display
7.mirror
enter u r choice : 1
enter total number of nodes:4
enter tree value: 2 3 4 6
2 3 4 6

1.create
2.find
3.find_min
```

```
4.find_max

5.longest_path

6.display

7.mirror

enter u r choice : 2

enter node to be searched 3

node found3

1.create

2.find

3.find_min

4.find_max

5.longest_path

6.display

7.mirror

enter u r choice : 3

The minimum value = 2

1.create

2.find

3.find_min

4.find_max

5.longest_path

6.display

7.mirror

enter u r choice : 4
```

```
The maximum value = 6
1.create
2.find
3.find_min
4.find_max
5.longest_path
6.display
7.mirror
enter u r choice : 5
longest path in tree  4
1.create
2.find
3.find_min
4.find_max
5.longest_path
6.display
7.mirror
enter u r choice : 6
2     3     4     6
1.create
2.find
3.find_min
4.find_max
5.longest_path
```

```
6.display

7.mirror

enter u r choice : 7

1.create

2.find

3.find_min

4.find_max

5.longest_path

6.display

7.mirror

enter u r choice :
```