Dsa-Lab-Task : 10

Roll No: 24P-0706 **Dept: BS-CS**

Name: Aazan Noor Khuwaja **Section : 3D**

Qns1:

```
#include<iostream>
using namespace std;

class node{
   public:
   int data;
   node *left,*right;
   node(int d):data(d){
      left=right=NULL;
   }
};
class BST{
private:
node* find_maximum(node* r)
{
      if(r->right!=NULL) r=find_maximum(r->right);
      return r;
}
node* find_minimum(node* r)
{
```

```
        if(r->left!=NULL) r=find_minimum(r->left);
        return r;
}
node* insert(node* r,int val)
   {
        if(r==NULL)return new node(val);
        else if(val < r->data)
        {
           r->left = insert(r->left,val);
        }
        else if(val > r->data)
        {
           r->right = insert(r->right,val);
        }
        return r;
   }
void inorder(node *t)
{
   if(t==NULL)return;

   inorder(t->left);
   cout<<t->data<<",";
   inorder(t->right);

}
public:
   node *root;
```

```
   BST()
   {
      root=NULL;
   }

void create_insert_tree()
{
   int d;
cout<<" Enter the value:"<<endl;
cin>>d;
  root=insert(root,d);

}
void inorder_traversal(){
   inorder(root);
}

void max_show()
{
   node*max=find_maximum(root);
   cout<<" The MAX value is:"<<max->data<<endl;
}
void min_show()
{
   node*min=find_minimum(root);
   cout<<" The MIN value is:"<<min->data<<endl;
}
```

```
};
int main()
{
   BST tree;
   tree.create_insert_tree();
   tree.create_insert_tree();
    tree.create_insert_tree();
     tree.create_insert_tree();
      tree.create_insert_tree();
       tree.create_insert_tree();
        tree.create_insert_tree();
   tree.inorder_traversal();
   cout<<endl;
   tree.max_show();
   tree.min_show();
}
```

Qns 2:

```
#include<iostream>
using namespace std;

class node{
   public:
   int data;
   node *left,*right;
```

```
    node(int d):data(d){
        left=right=NULL;
    }
};
class BST {
private:

node *root;
node* insert(node* r,int val)
    {
        if(r==NULL)return new node(val);
        else if(val<r->data)
        {
            r->left=insert(r->left,val);
        }
        else if(val>r->data)
        {
            r->right=insert(r->right,val);
        }
        return r;
    }

void inorder(node *t)
{
    if(t==NULL) return ;
    inorder(t->left);
    cout<<t->data<<" ";
    inorder(t->right);
}
```

```
int count_nodes(node *k)
{
   if(k==NULL)return 0;
   return 1+ count_nodes(k->right)+count_nodes(k->left);
}

public:

   BST()
   {
      root=NULL;
   }

void create_insert_tree()
{
   int d;
cout<<" Enter the value:"<<endl;
cin>>d;
  root=insert(root,d);

}
void inorder_traversal(){
   inorder(root);
}

void cheak_nodes_both_sides()
{
   int right_count=count_nodes(root->right);
   cout<<right_count<<endl;
```

```
    int left_count=count_nodes(root->left);
     cout<<left_count<<endl;
    if(right_count==left_count)
    {
       cout<<" Both side childrens are equal!"<<endl;
    }
    else{
       cout<<" Childrens are not equal"<<endl;
    }
}


};
int main()
{
    BST tree;
    int choose;
    while(true)
    {
        cout <<"enter the value to choose : \n1.Insert In Tree\n2.Inorder Traversal\n3.Cheak Equal Childs
or not\n4. exit...."<<endl;
        cin>>choose;
       switch(choose)
       {
          case 1:
             tree.create_insert_tree();
             break;
          case 2:
             tree.inorder_traversal();
             break;
```

```
            case 3:
               tree.cheak_nodes_both_sides();
               break;
            case 4:
               exit(0);
               break;
            default:
               cout<<"Not a valid input!"<<endl;
               break;
         }
      }
}
```

Qns 3:

```
#include<iostream>
using namespace std;

class node{
   public:
   int data;
   node *left,*right;
   node(int d):data(d){
      left=right=NULL;
   }
};
class BST {
private:

node *root;
```

```
node* insert(node* r,int val)
   {
      if(r==NULL)return new node(val);
      else if(val<r->data)
      {
         r->left=insert(r->left,val);
      }
      else if(val>r->data)
      {
         r->right=insert(r->right,val);
      }
      return r;
   }

void inorder(node *t)
{
   if(t==NULL) return ;
   inorder(t->left);
   cout<<t->data<<" ";
   inorder(t->right);
}
int count_nodes(node *k)
{
   if(k==NULL)return 0;
   return 1+ count_nodes(k->right)+count_nodes(k->left);
}
node *right_most_left_subtree(node* n)
{
   while(n->right!=NULL){
```

```
    n=n->left;
    }
    return n;
}
node *left_most_right_subtree(node* n)
{
    while(n->left!=NULL){
    n=n->right;
    }
    return n;
}
void get_pre_succ(node *root,int target)
{
    node *pre,*succ;
    node* current=root;
    pre=succ=NULL;
    while(current!=NULL)
    {
        if(target<current->data)
        {
            succ=current;
            current=current->left;
        }
        else if(target>current->data)
        {
            pre=current;
            current=current->right;
        }
        else{
```

```
        if(current->left!=NULL)
        {
           pre=right_most_left_subtree(current->left);
        }
        if(current->right!=NULL)
        {
           succ=left_most_right_subtree(current->right);
        }
        break;
      }

   }

   if (pre != NULL)
      cout<<"The Predecessor is:"<<pre->data<<endl;
   else
      cout << "No Predecessor exist"<<endl;
   if (succ != NULL)
      cout << "The Successor is:"<<succ->data <<endl;
   else
      cout<< "No Successor exist" << endl;

}

public:

   BST()
   {
      root=NULL;
```

```
    }

void create_insert_tree()
{
    int d;
cout<<" Enter the value:"<<endl;
cin>>d;
   root=insert(root,d);

}
void inorder_traversal(){
    inorder(root);
}


void cheak_nodes_both_sides()
{
    int right_count=count_nodes(root->right);
    cout<<right_count<<endl;
    int left_count=count_nodes(root->left);
     cout<<left_count<<endl;
    if(right_count==left_count)
    {
        cout<<" Both side childrens are equal!"<<endl;
    }
    else{
        cout<<" Childrens are not equal"<<endl;
    }
}
```

```
void pre_and_succ_target()
{
   int n;
   cout<<"Enter the target :"<<endl;
   cin>>n;
   get_pre_succ(root,n);
}




};
int main()
{
   BST tree;
   int choose;
   while(true)
   {
      cout<<endl;
       cout <<"enter the value to choose : \n1.Insert In Tree\n2.Inorder Traversal\n3.Cheak Equal Childs or not\n4.Find Successor and Predecessor\n5. exit...."<<endl;
       cin>>choose;
      switch(choose)
      {
         case 1:
            tree.create_insert_tree();
            break;
         case 2:
            tree.inorder_traversal();
```

```
            break;
        case 3:
            tree.cheak_nodes_both_sides();
            break;
        case 4:
            tree.pre_and_succ_target();
            break;
        case 5:
            exit(0);
            break;
        default:
            cout<<"Not a valid input!"<<endl;
            break;
      }
   }
```

Qns 4:

```
#include<iostream>
using namespace std;

class node{
   public:
   int data;
   node *left,*right;
   node(int d):data(d){
      left=right=NULL;
   }
};
```

```
class BST{
private:
node* insert(node* r,int val)
{
   if(r==NULL)return new node(val);
   else if(val<r->data)
   {
      r->left=insert(r->left,val);
   }
   else if(val>r->data)
   {
      r->right=insert(r->right,val);
   }
   return r;
}
void inorder(node *t)
{
   if(t==NULL)return;
   inorder(t->left);
   cout<<t->data<<",";
   inorder(t->right);
}
node* insert_after_val(node* r, int val)
{
   if(r==NULL)return new node(val);
   if(val<r->data)r->left=insert_after_val(r->left,val);
   else if(val>r->data)r->right=insert_after_val(r->right,val);
   return r;
}
```

```
node* insert_after_target(node* r,int target,int new_val)
{
   if(r==NULL)return NULL;
   if(r->data==target)
   {
      if(r->right==NULL)r->right=new node(new_val);
      else r->right=insert_after_val(r->right,new_val);
      return r;
   }
   if(target<r->data)r->left=insert_after_target(r->left,target,new_val);
   else r->right=insert_after_target(r->right,target,new_val);
   return r;
}
public:
   node *root;
   BST()
   {
      root=NULL;
   }
void create_insert_tree()
{
   int d;
cout<<" Enter the value:"<<endl;
cin>>d;
  root=insert(root,d);
}
void inorder_traversal(){
   inorder(root);
}
```

```
void insert_after()
{
   int target,new_val;
   cout<<"Enter target and new value:"<<endl;
   cin>>target>>new_val;
   root=insert_after_target(root,target,new_val);
}
};
int main()
{
   BST tree;
   tree.create_insert_tree();
    tree.create_insert_tree();
     tree.create_insert_tree();
      tree.create_insert_tree();
       tree.create_insert_tree();
        tree.create_insert_tree();
         tree.create_insert_tree();
   tree.inorder_traversal();
   cout<<endl;
   tree.insert_after();
   tree.inorder_traversal();
}
```