Assignment 3

<u>Problem Statement :</u> Implement Binary Search Tree and perform following operations:(Non recursive implementation)

a. Insert
b. Search
c. Mirror Image
d. Display In-order traversal
e. Display level wise
f. Delete

<u>Code :</u>

```
#include<iostream>
using namespace std;
struct node
{
  int data;
  node *left, *right;
};

class stack
{
  public:
    node *st[20];
    int data,top;
  public:
    stack()
    {
      top = -1;
    }
    int isEmpty()
    {
      if(top==-1)
        return 1;
      else
        return 0;
    }
    void push(node *nwnode)
    {
      top++;
      st[top] = nwnode;
    }
    node *pop()
    {
      node *nwnode;
      nwnode = st[top];
      top--;
      return(nwnode);
    }
};

class queue
{
  node *que[20];
  int data,rear,front;
  public:
    queue()
    {
      rear = front = -1;
    }
    int isEmpty()
    {
      if(rear==front)
        return 1;
      else
        return 0;
    }
    int isFull()
    {
      if(rear==20)
        return 1;
      else
        return 0;
    }
    void add(node *nwnode)
    {
      if(isFull())
        cout<<"\nQueue Overflow";
      else
      {
        rear++;
        que[rear] = nwnode;
      }
    }
    node *del()
    {
      node *nwnode;
      if(isEmpty())
      {
        cout<<"\nQueue is Empty";
      }
      else
      {
        front++;
        nwnode = que[front];
        return(nwnode);
      }
    }
};

class bst
{
  public:
  node *root,*mirror;
  bst()
  {
    root = NULL;
    mirror = NULL;
  }
  void create();
  void insert();
  void search();
  void treecopy();
  void mirror_image();
  void inorder(node *);
  void bfs();
  void delete_node();
};

void bst::create()
{
  int data;
  char ch;
  node *temp = new node;
  if(root==NULL)
  {
    root = new node;
    cout<<"\nEnter the value of root : ";
    cin>>data;
    root->data=data;
    root->left = root->right = NULL;
  }
  else
  {
    temp = root;
    cout<<"\nEnter the value of node : ";
    cin>>data;
    while(1)
    {
      if(data<temp->data)
      {
        if(temp->left!=NULL)
        {
          temp = temp->left;
        }
        else
        {
          temp->left = new node;
          temp = temp->left;
          temp->data = data;
          temp->left = temp->right = NULL;
          break;
        }
      }
      if(data>temp->data)
      {
        if(temp->right!=NULL)
        {
          temp = temp->right;
        }
        else
        {
          temp->right = new node;
          temp = temp->right;
          temp->data = data;
          temp->left = temp->right = NULL;
          break;
        }
      }
      if(data==temp->data)
        break;
    }
  }
}

void bst::insert()
{
  int n;
  cout<<"\nEnter number of nodes : ";
  cin>>n;
  for(int i=0;i<n;i++)
  {
    create();
  }
}

void bst::treecopy()
{
  stack S,S1;
  node *temp;
  node *temp1;
  temp = root;
  mirror = new node;
  temp1  = mirror;
  temp1->left = temp1->right = NULL;
  while(1)
  {
    temp1->data = temp->data;
    if(temp->right!=NULL)
    {
      S.push(temp->right);
      temp1->right = new node;
      temp1->right->left = temp1->right->right = NULL;
      S1.push(temp1->right);
    }
    if(temp->left!=NULL)
    {
      S.push(temp->left);
      temp1->left = new node;
      temp1->left->left = temp1->left->right = NULL;
      S1.push(temp1->left);
    }
    if(S.isEmpty())
      break;
    temp = S.pop();
    temp1 = S1.pop();
  }
}

void bst::mirror_image()
{
  stack S;
  node *tmp;
  node *cur;
  cur = mirror;
  while(1)
  {
    while(cur!=NULL)
    {
      tmp = cur->left;
      cur->left = cur->right;
      cur->right = tmp;
      S.push(cur);
      cur = cur->left;
    }
    if(S.isEmpty())
      break;
    cur = S.pop();
    cur = cur->right;
  }
}

void bst::search()
{
  int value,flag=0,z=0;
  node *temp = new node;
  temp = root;
  cout<<"\nEnter the data to search : ";
  cin>>value;
  while(1)
  {
    if(value==temp->data)
    {
      flag=1;
      break;
    }
    else if(value<temp->data)
    {
      if(temp->left==NULL)
        break;
      temp = temp->left;
      z++;
    }
    else if(value>temp->data)
    {
      if(temp->right==NULL)
        break;
      temp = temp->right;
      z++;
    }
  }
  if(flag==1)
  {
    cout<<"\ndata Found at Level "<<z;
  }
  else{
    cout<<"\ndata not Found";
  }
}

void bst::inorder(node *temp)
{
  stack S;
  while(1)
  {
    while(temp!=NULL)
    {
      S.push(temp);
      temp = temp->left;
    }
    if(S.isEmpty())
      return;
    temp = S.pop();
    cout<<temp->data<<" ";
    temp = temp->right;
  }
}

void bst::bfs()
{
  queue Q;
  node *nwnode = new node;
  nwnode = root;
  while(1)
  {
    cout<<nwnode->data<<" ";
    if(nwnode->left!=NULL)
      Q.add(nwnode->left);
    if(nwnode->right!=NULL)
      Q.add(nwnode->right);
    if(Q.isEmpty())
      break;
    nwnode = Q.del();
  }
}

void bst::delete_node()
{
```

```
node *temp,*prev,*x,*l,*r;
temp = root;
int deldata, flag=0, status = 0;
cout<<"\nEnter the data to delete : ";
cin>>deldata;
//Searching for the entered data
//status will check whether the parent node has the target leaf node as left or right
//so that it can become NULL after deletion
while(1)
{
  if(deldata==temp->data)
  {
    flag=1;
    break;
  }
  else if(deldata<temp->data)
  {
    if(temp->left==NULL)
      break;
    prev = temp;
    temp = temp->left;
    status = 1;
  }
  else if(deldata>temp->data)
  {
    if(temp->right==NULL)
      break;
    prev = temp;
    temp = temp->right;
    status = 2;
  }
}
if(flag==1)
{
  if(temp->left==NULL && temp->right==NULL)
  {
    if(status==1)
      prev->left = NULL;
    if(status==2)
      prev->right = NULL;
    delete temp;
  }
  if(temp->left!=NULL && temp->right==NULL)
  {
    x = temp->left;
    prev->left = x;
    delete temp;
  }
  if(temp->left==NULL && temp->right!=NULL)
  {
    x = temp->right;
    prev->right = x;
    delete temp;
  }
  if(temp->left!=NULL && temp->right!=NULL)
  {
    l = temp->left;
    r = temp->right;
    prev->right=r;
    if(l->data<r->data)
    {
      if(r->left!=NULL)
      {
        r = r->left;
        }
        else{
          r->left = l;
        }
      }
      if(l->data>r->data)
      {
        if(r->right!=NULL)
        {
          r = r->right;
        }
        else{
          r->right = l;
        }
      }
      delete temp;
    }
  }
  else{
    cout<<"\nData not found!!!";
  }
}


int main()
{
  bst obj;
  int choice,ch;
  cout<<"\n1. Create a Binary Search Tree \n2. Exit"<<endl;
  cout<<"\nEnter your choice : ";
  cin>>choice;
  cout<<"\n------------------------------------------------------------------";
  if(choice==1)
  {
    obj.insert();
    cout<<"\nTree Created Successfully!!!";
    while(1)
    {
      cout<<"\n------------------------------------------------------------------";
      cout<<"\n1. Insert node \n2. Search \n3. Mirror Image \n4. Inorder Traversal \n5. Level wise Traversal \n6.
Delete \n7. Exit";
      cout<<"\nEnter your choice : ";
      cin>>ch;
      cout<<"\n------------------------------------------------------------------";
      if(ch==1)
      {
        obj.insert();
        cout<<"\nInsertion successfull";
      }
      else if(ch==2)
      {
        obj.search();
      }
      else if(ch==3)
      {
        obj.treecopy();
        obj.mirror_image();
        cout<<"\nMirror Image Created Successfully!!!";
        cout<<"\nInorder Traversal of Mirror Image of tree : ";
        obj.inorder(obj.mirror);
      }
      else if(ch==4)
      {
        cout<<"\nInorder Traversal : ";
        obj.inorder(obj.root);
      }
      else if(ch==5)
      {
        cout<<"\nLevel wise Traversal : ";
        obj.bfs();
      }
      else if(ch==6)
      {
        obj.delete_node();
        cout<<"\nNode deleted Successfully";
      }
      else
      {
        cout<<"\nProgram Exited!!!";
        break;
      }
    }
  }
  else
  {
    cout<<"\nProgram Exited!!!";
  }
}
```

OUTPUT

```
C:\Users\Safir\Desktop\ADS>g++ 3_binary_search_tree.cpp
C:\Users\Safir\Desktop\ADS>a
1. Create a Binary Search Tree
2. Exit
Enter your choice : 1
--------------------------------------------------------------------
Enter number of nodes : 5
Enter the value of root : 4
Enter the value of node : 2
Enter the value of node : 6
Enter the value of node : 5
Enter the value of node : 7
Tree Created Successfully!!!
--------------------------------------------------------------------
1. Insert node
2. Search
3. Mirror Image
4. Inorder Traversal
5. Level wise Traversal
6. Delete
7. Exit
Enter your choice : 2
--------------------------------------------------------------------
Enter the data to search : 5
data Found at Level 2
--------------------------------------------------------------------
1. Insert node
2. Search
3. Mirror Image
4. Inorder Traversal
5. Level wise Traversal
6. Delete
7. Exit
Enter your choice : 3
--------------------------------------------------------------------
Mirror Image Created Successfully!!!
Inorder Traversal of Mirror Image of tree : 7 6 5 4 2
--------------------------------------------------------------------
1. Insert node
2. Search
3. Mirror Image
4. Inorder Traversal
5. Level wise Traversal
6. Delete
7. Exit
Enter your choice : 4
--------------------------------------------------------------------
 Inorder Traversal : 2 4 5 6 7
--------------------------------------------------------------------
1. Insert node
2. Search
3. Mirror Image
4. Inorder Traversal
5. Level wise Traversal
6. Delete
```

```
7. Exit
Enter your choice : 5
------------------------------------------------------------------
Level wise Traversal : 4 2 6 5 7
------------------------------------------------------------------
1. Insert node
2. Search
3. Mirror Image
4. Inorder Traversal
5. Level wise Traversal
6. Delete
7. Exit
Enter your choice : 6
------------------------------------------------------------------
Enter the data to delete : 6
Node deleted Successfully
------------------------------------------------------------------
1. Insert node
2. Search
3. Mirror Image
4. Inorder Traversal
5. Level wise Traversal
6. Delete
7. Exit
Enter your choice : 4
------------------------------------------------------------------
Inorder Traversal : 2 4 5 7
------------------------------------------------------------------
1. Insert node
2. Search
3. Mirror Image
4. Inorder Traversal
5. Level wise Traversal
6. Delete
7. Exit
Enter your choice : 7
------------------------------------------------------------------
Program Exited!!!
```