NAME – AARYAN BAIRAGI

ROLL NO – 27002

CLASS – SE IT B

/\*Implement binary search Tree and perform following operations:

a. Insert

b. Delete

c. Search

d. Mirror image

e. Display

f. Display level wise

\*/

#include<iostream>

using namespace std;

typedef struct TreeNode

{

    int data;

    struct TreeNode \*left;

    struct TreeNode \*right;

}TreeNode;

typedef struct Node

{

   struct TreeNode \*TreeNode;

   struct Node \*next;

}Node;

class Queue

{

   Node \*front,\*rear;

public:

    Queue()

    {

       front=NULL;

       rear=NULL;

    }

    int isEmpty()

    {

       return (front==NULL)?1:0;

    }

    void enque(TreeNode \*t)

    {

        Node \*p;

        p=new Node();

        p->TreeNode = t;

        p->next=NULL;

        if(front==NULL)

        {

         front=p;

         rear=p;

        }

        else

        {

         rear->next = p;

         rear = rear->next;

        }

    }

    TreeNode\* deque()

    {

        Node \*p;

        TreeNode \*temp;

        p = front;

        temp = front->TreeNode;

        if(front==rear)

        {

            front=NULL;

            rear=NULL;

        }

        else

        {

            front = front->next;

        }

        delete p;

        return temp;

    }

};

class Tree

{

    TreeNode \*t;

public:

    Tree()

    {

        t=NULL;

    }

    TreeNode \*insert(int x)

    {

        TreeNode \*p,\*q,\*r;

        p = new TreeNode();

        p->data=x;

        p->left=NULL;

        p->right=NULL;

        if(t==NULL)

            return p;

        q=t;

        r=t;

        while(r!=NULL)

        {

            q=r;

            if(x < r->data)

                r=r->left;

            else

                r=r->right;

        }

        if(x<q->data)

            q->left=p;

        else

            q->right=p;

        return t;

    }

    TreeNode \*create()

    {

        int n,i,key;

        cout<<"\nEnter the number of Nodes : ";

        cin>>n;

        for(i=0;i<n;i++)

        {

            cout<<"\nEnter the data : ";

            cin>>key;

            t=insert(key);

        }

        return t;

    }

    void inorder(TreeNode \*t)

    {

        if(t!=NULL)

        {

            inorder(t->left);

            cout<<"\t"<<t->data;

            inorder(t->right);

        }

    }

    TreeNode\* search(int key)

    {

        TreeNode \*s=t;

        while(s!=NULL)

        {

            if(s->data==key)

                return t;

            else if(s->data<key)

                s=s->right;

            else

                s=s->left;

        }

        return NULL;

    }

    TreeNode \*getMin(TreeNode \*r)

    {

        while(r->left != NULL)

        {

            r=r->left;

        }

        return r;

    }

    TreeNode \*del(TreeNode \*t,int key)

    {

        TreeNode \*temp;

        if(t==NULL)

        {

            return NULL;

        }

        if(key<t->data)

        {

            t->left=del(t->left,key);

            return t;

        }

        if(key>t->data)

        {

            t->right=del(t->right,key);

            return t;

        }

        //element found

        //no child

        if(t->left==NULL&t->right==NULL)

        {

            temp=t;

            delete temp;

            return NULL;

        }

        //one child

        if(t->left!=NULL&&t->right==NULL)

        {

            temp=t;

            t=t->left;

            delete temp;

            return t;

        }

        if(t->left==NULL&&t->right!=NULL)

        {

            temp=t;

            t=t->right;

            delete temp;

            return t;

        }

        //both child present

        temp=getMin(t->right);

        t->data=temp->data;

        t->right=del(t->right,temp->data);

        return t;

    }

    TreeNode \*mirror(TreeNode \*t)

    {

        TreeNode \*temp;

        if(t==NULL)

        {

            return NULL;

        }

        temp=t->left;

        t->left=mirror(t->right);

        t->right=mirror(temp);

        return t;

    }

    void level\_wise()

    {

        TreeNode \*t1;

        Queue q1;

        if(t==NULL)

            return;

        q1.enque(t);

        cout<<"\n "<<t->data;

        while(q1.isEmpty()!=1)

        {

            cout<<"\n";

            Queue q2;

            while(q1.isEmpty()!=1)

            {

                t1=q1.deque();

                if(t1->left!=NULL)

                {

                    q2.enque(t1->left);

                    cout<<" "<<t1->left->data;

                }

                if(t1->right!=NULL)

                {

                    q2.enque(t1->right);

                    cout<<" "<<t1->right->data;

                }

            }

            q1=q2;

        }

    }

};

int main()

{

    int choice,key;

    TreeNode \*root,\*result;

    Tree t;

    do

    {

        cout<<"\n=======================================";

        cout<<"\nMENU";

        cout<<"\n=======================================";

        cout<<"\n1.Create\n2.Insert\n3.Display\n4.Search\n5.Delete\n6.Mirror image\n7.Display Level wise\n8.Minimum\n9.Exit";

        cout<<"\n\nEnter your Choice : ";

        cin>>choice;

        switch(choice)

        {

        case 1:root = t.create();break;

        case 2:

            cout<<"\nEnter the number to Insert : ";

            cin>>key;

            root=t.insert(key);

            break;

        case 3:

            cout<<"\n-----------------------------------------------";

            cout<<"\nBINARY TREE :-";

            t.inorder(root);

            cout<<"\n-----------------------------------------------";

            break;

        case 4:

            cout<<"\n--------------------------";

            cout<<"\nEnter the Node to search :";

            cin>>key;

            result = t.search(key);

            if(result==NULL)

            {

                cout<<"\nNODE "<<key<<" NOT FOUND";

            }

            else

            {

                cout<<"\nNODE "<<key<<" IS FOUND";

            }

            cout<<"\n--------------------------";

            break;

        case 5:

            cout<<"\n--------------------------";

            cout<<"\nEnter the Node to delete :";

            cin>>key;

            result = t.del(root,key);

            root=result;

            cout<<"\nNODE DELETED !!";

            cout<<"\n--------------------------";

            break;

        case 6:

            cout<<"\n--------------------------";

            root=t.mirror(root);

            cout<<"\nMirror image of the binary Tree :";

            t.inorder(root);

            cout<<"\n--------------------------";

            break;

        case 7:

            cout<<"\n--------------------------";

            cout<<"\nLevel Wise Display :";

            cout<<"\n--------------------------";

            t.level\_wise();

            cout<<"\n--------------------------";

            break;

        case 8:

            result = t.getMin(root);

            cout<<"\n--------------------------";

            cout<<"\nMINIMUM = "<<result->data;

            cout<<"\n--------------------------";

            break;

        case 9:return 0;

        default:cout<<"\nInvalid Choice !!";

        }

    }while(choice!=9);

}