/\*

**Name: Krushabhsingh B. Suryawanshi**

**Roll: ITSEB606**

ASSIGNMENT NO:05

AIM:-

Implement binary search tree and perform following operations:

a)  Insert (Handle insertion of duplicate entry)

b)  Delete

c)  Search

d)  Display tree (Traversal)

e)  Display - Depth of tree

f)  Display - Mirror image

g)  Create a copy

h)  Display all parent nodes with their child nodes

i)  Display leaf nodes

j)  Display tree level wise

(Note: Insertion, Deletion, Search and Traversal are compulsory, from rest of operations, perform Any three) \*/

#include <iostream>

using namespace **std**;

struct **tnode**{

    int data;

**tnode**\* left;

**tnode**\* right;

};

struct **qnode**{

**tnode**\* qdata;

**qnode**\* next;

};

class **Queue**{

**qnode**\* front;

**qnode**\* rear;

    public:

**Queue**()

    {

        front=rear=**NULL**;

    }

    int **isempty**()

    {

        if(front==**NULL**)

         return 1;

        return 0;

    }

    void **enqueue**(**tnode**\* t)

    {

**qnode**\* temp;

        temp=new **qnode**();

        temp->qdata=t;

        temp->next=**NULL**;

        if(front==**NULL**)

        {

            front=temp;

            rear=temp;

        }

        else

        {

            rear->next=temp;

            rear=rear->next;

        }

    }

**tnode**\* **dequeue**()

    {

**tnode**\* temp;

**qnode**\* p;

        p=front;

        temp=front->qdata;

        if(front==rear)

        {

            front=rear=**NULL**;

        }

        else

        {

            front=front->next;

        }

        delete(p);

        return temp;

    }

};

class **Tree**{

**tnode**\* t;

    public:

**Tree**()

        {

          t=**NULL**;

        }

**tnode**\* **insert**(int key)

               {

**tnode** \*temp,\*q,\*r;

           temp=new **tnode**();

           temp->data=key;

           temp->left=temp->right=**NULL**;

           if(t==**NULL**)

             return temp;

           q=t;

           r=t;

           while(r!=**NULL**)

          {

           q=r;

           if(key<r->data)

             r=r->left;

           else

            r=r->right;

              }

             if(key<q->data)

              q->left=temp;

            else

             q->right=temp;

            return t;

            }

**tnode**\* **create**()

       {

         int n,key;

         cout**<<**"enter the number of nodes\n";

         cin**>>**n;

         for(int i=0;i<n;i++)

         {

           cout**<<**"enter the data\n";

           cin**>>**key;

           t=**insert**(key);

         }

         return t;

       }

       void **inorder**(**tnode**\* t)

       {

        if(t==**NULL**)

         return;

**inorder**(t->left);

        cout**<<**t->data**<<**"\n";

**inorder**(t->right);

     }

**tnode**\* **search**(int key)

     {

**tnode**\* s=t;

        while(s!=**NULL**)

        {

        if(s->data==key)

         return t;

*//return s;*

        else if(s->data<key)

         s=s->right;

        else

         s=s->left;

            }

        return **NULL**;

       }

**tnode**\* **find\_min**(**tnode**\* t)

       {

        while(t->left!=**NULL**)

        {

         t=t->left;

             }

        return t;

       }

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

       {

**tnode**\* temp;

        if(t==**NULL**)

         return **NULL**;

        if(t->data<key)

        {

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

         return t;

            }

            if(t->data>key)

            {

            t->left=**del**(t->left,key);

            return t;

        }

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

        {

         temp=t;

         delete(temp);

         return **NULL**;

        }

        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;

        }

        temp=**find\_min**(t->right);

        t->data=temp->data;

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

        return t;

       }

**tnode**\* **mirror**(**tnode**\* t)

       {

**tnode**\* temp;

        if(t==**NULL**)

         return **NULL**;

        temp=t->left;

        t->left=**mirror**(t->right);

        t->right=**mirror**(temp);

        return t;

       }

       void **leaf**(**tnode**\* t)

       {

        if(t==**NULL**)

         return;

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

         cout**<<endl<<**t->data**<<**"\t";

**leaf**(t->left);

**leaf**(t->right);

       }

**tnode**\* **copy**(**tnode**\* t)

       {

**tnode**\* p;

        p=**NULL**;

        if(t!=**NULL**)

        {

            p=new **tnode**();

            p->data=t->data;

            p->left=**copy**(t->left);

            p->right=**copy**(t->right);

        }

         return p;

       }

       int **height**(**tnode**\* t)

       {

         int hl,hr;

         if(t==**NULL**)

          return 0;

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

          return 0;

         hl=**height**(t->left);

         hr=**height**(t->right);

         if(hl>hr)

          return hl+1;

         else

          return hr+1;

       }

       void **levelwise**()

       {

**tnode**\* t1;

**Queue** q1;

        if(t==**NULL**)

         return;

        q1.**enqueue**(t);

        cout**<<**"\n"**<<**t->data;

        while(q1.**isempty**()!=1)

        {

         cout**<<endl**;

**Queue** q2;

         while(q1.**isempty**()!=1)

         {

              t1=q1.**dequeue**();

          if(t1->left!=**NULL**)

          {

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

           q2.**enqueue**(t1->left);

         }

         if(t1->right!=**NULL**)

         {

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

            q2.**enqueue**(t1->right);

         }

        }

         q1**=**q2;

        }

        }

       void **parent**(**tnode**\* t)

       {

        if(t==**NULL**)

         return;

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

            {

         cout**<<**"\t"**<<**t->data;

         cout**<<**"\t"**<<**t->left->data;

         cout**<<endl**;

        }

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

        {

            cout**<<**"\t"**<<**t->data;

            cout**<<**"\t"**<<**t->right->data;

            cout**<<endl**;

        }

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

        {

            cout**<<**"\t"**<<**t->data;

            cout**<<**"\t"**<<**t->left->data**<<**"\t"**<<**t->right->data**<<endl**;

            cout**<<endl**;

        }

**parent**(t->left);

**parent**(t->right);

       }

};

int **main**()

{

    int ch,key,cnt;

**Tree** t;

**tnode** \*root,\*rt,\*result;

    do{

        cout**<<endl**;

        cout**<<**"1)Create \n2)Insert \n3)Display \n4)Search \n5)Delete \n6)Mirror Image \n7)Create Copy \n8)Find depth \n";

        cout**<<**"9)Minimum \n10)Display tree level wise \n11)Display leaf nodes \n 12)Display parent node with child node\n ";

        cout**<<**"13)Exit \n";

        cout**<<**"enter your choice \n";

        cin**>>**ch;

        switch(ch)

        {

            case 1:

                root=t.**create**();

                break;

            case 2:

                cout**<<**"enter the number to insert \n";

                cin**>>**key;

                root=t.**insert**(key);

                break;

            case 3:

                cout**<<**"Inorder display\n";

                t.**inorder**(root);

                cout**<<endl**;

                break;

            case 4:

                cout**<<**"enter the number to search\n";

                cin**>>**key;

                rt=t.**search**(key);

                if(rt==**NULL**)

                 cout**<<**"element "**<<** key **<<**" not found \n";

                else

                 cout**<<**"element "**<<** key **<<**" found\n";

                 break;

            case 5:

                cout**<<**"enter the node to delete \n";

                cin**>>**key;

                result=t.**del**(root,key);

                root=result;

                cout**<<**"element deleted successfully \n";

                break;

            case 6:

                rt=t.**mirror**(root);

                cout**<<**"\n mirror image of binary tree is:-"**<<endl**;

                t.**inorder**(rt);

                break;

            case 7:

                cout**<<**"copied tree\n";

                rt=t.**copy**(root);

                t.**inorder**(rt);

                break;

            case 8:

                cnt=t.**height**(root);

                cout**<<**"height of tree :"**<<**cnt**<<endl**;

                break;

            case 9:

                result=t.**find\_min**(root);

                cout**<<**"the minimum element is "**<<**result->data**<<endl**;

                break;

            case 10:

                cout**<<**"displaying nodes levelwise"**<<endl**;

                t.**levelwise**();

                break;

            case 11:

                cout**<<**"leaf nodes are:"**<<endl**;

                t.**leaf**(root);

                break;

            case 12:

                cout**<<**"the parent node with child node are: "**<<endl**;

                t.**parent**(root);

                break;

            case 13:

                break;

            default:

                cout**<<**"Enter a valid choice!"**<<endl**;

        }

    }while(ch!=13);

}

// :Output:

G:\it\_se>cd "g:\it\_se\binary-search-tree-\" && g++ binary\_search\_tree.cpp -o binary\_search\_tree && "g:\it\_se\binary-search-tree-\"binary\_search\_tree

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

1

enter the number of nodes

3

enter the data

15

enter the data

16

enter the data

17

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

3

Inorder display

15

16

17

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

2

enter the number to insert

14

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

3

Inorder display

14

15

16

17

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

4

enter the number to search

17

element 17 found

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

5

enter the node to delete

17

element deleted successfully

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

3

Inorder display

14

15

16

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

6

mirror image of binary tree is:-

16

15

14

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

7

copied tree

16

15

14

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

3

Inorder display

16

15

14

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

8

height of tree :1

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

9

the minimum element is 16

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

10

displaying nodes levelwise

15

16 14

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

11

leaf nodes are:

16

14

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

12

the parent node with child node are:

15 16 14

1)Create

2)Insert

3)Display

4)Search

5)Delete

6)Mirror Image

7)Create Copy

8)Find depth

9)Minimum

10)Display tree level wise

11)Display leaf nodes

12)Display parent node with child node

13)Exit

enter your choice

13

g:\it\_se\binary-search-tree->

