Group B

Practical 4

```
/*Write C++/Java program to draw 2-D object and
perform following basic transformations,

a) Scaling

b) Translation

c) Rotation

Use operator overloading. */

#include<iostream>

#include<stdlib.h>

#include<graphics.h>

#include<math.h>

using namespace std;

class POLYGON

{

    private:

        int
p[10][10],Trans_result[10][10],Trans_matrix[10]
[10];

        float
Rotation_result[10][10],Rotation_matrix[10][10]
;
```

```
        float
Scaling_result[10][10],Scaling_matrix[10][10];
        float
Shearing_result[10][10],Shearing_matrix[10][10]
;
        int
Reflection_result[10][10],Reflection_matrix[10]
[10];

    public:
    int accept_poly(int [][10]);
    void draw_poly(int [][10],int);
    void draw_polyfloat(float [][10],int);
    void matmult(int [][10],int
[][10],int,int,int,int [][10]);
    void matmultfloat(float [][10],int
[][10],int,int,int,float [][10]);
    void shearing(int [][10],int);
    void scaling(int [][10],int);
    void rotation(int [][10],int);
    void translation(int [][10],int);
    void reflection(int [][10],int);
};

int POLYGON :: accept_poly(int p[][10])
{
    int i,n;
    cout<<"\n\nEnter number of vertices : ";
    cin>>n;
    for(i=0;i<n;i++)
    {
        cout<<"\n\nEnter (x,y) Co-ordinate of
point P"<<i<<" : ";
        cin >> p[i][0] >> p[i][1];
        p[i][2] = 1;
    }

    for(i=0;i<n;i++)
    {
        cout<<"\n";
        for(int j=0;j<3;j++)
        {
            cout<<p[i][j]<<"\t\t";
        }
    }

    return n;
}
void POLYGON :: draw_poly(int p[][10], int n)
{
    int i,gd = DETECT,gm;
    initgraph(&gd,&gm,NULL);
    line(320,0,320,480);
    line(0,240,640,240);

    for(i=0;i<n;i++)
    {
        if(i<n-1)
        {
            line(p[i][0]+320, -p[i][1]+240,
p[i+1][0]+320, -p[i+1][1]+240);

        }
        else
            line(p[i][0]+320, -p[i][1]+240,
p[0][0]+320, -p[0][1]+240);
    }




}

void POLYGON :: draw_polyfloat(float p[][10],
int n)
{
    int i,gd = DETECT,gm;
    initgraph(&gd,&gm,NULL);
    line(320,0,320,480);
    line(0,240,640,240);


    for(i=0;i<n;i++)
    {
        if(i<n-1)
        {
            line(p[i][0]+320, -p[i][1]+240,
p[i+1][0]+320, -p[i+1][1]+240);


        }
        else
            line(p[i][0]+320, -p[i][1]+240,
p[0][0]+320, -p[0][1]+240);
    }



}



void POLYGON :: translation(int p[10][10],int
n)
{
    int tx,ty,i,j; int i1,j1,k1,r1,c1,c2;
         r1=n;c1=c2=3;
    cout << "\n\nEnter X-Translation tx : ";
    cin >> tx;
    cout << "\n\nEnter Y-Translation ty : ";
    cin >> ty;
    for(i=0;i<3;i++)
    for(j=0;j<3;j++)
        Trans_matrix[i][j] = 0;
    Trans_matrix[0][0] = Trans_matrix[1][1] =
Trans_matrix[2][2] = 1;
    Trans_matrix[2][0] = tx;
    Trans_matrix[2][1] = ty;

    for(i1=0;i1<10;i1++)
    for(j1=0;j1<10;j1++)
        Trans_result[i1][j1] = 0;
    for(i1=0;i1<r1;i1++)
    for(j1=0;j1<c2;j1++)
    for(k1=0;k1<c1;k1++)
        Trans_result[i1][j1] =
Trans_result[i1][j1]+(p[i1][k1] *
Trans_matrix[k1][j1]);
    cout << "\n\nPolygon after Translation : ";
    draw_poly(Trans_result,n);
}

void POLYGON :: rotation(int p[][10],int n)
{
    float type,Ang,Sinang,Cosang;
```

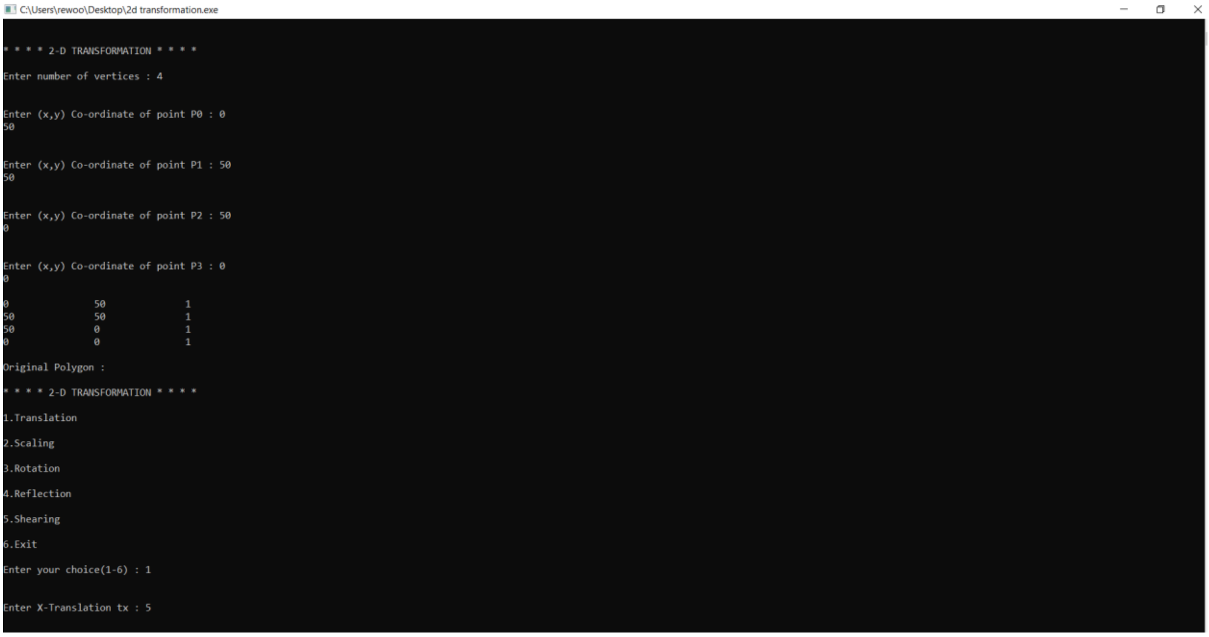
```
        int i,j; int i1,j1,k1,r1,c1,c2;
        r1=n;c1=c2=3;
    cout << "\n\nEnter the angle of rotation in
degrees : ";
    cin >> Ang;
    cout << "\n\n* * * * Rotation Types * * *
*";
    cout << "\n\n1.Clockwise Rotation
\n\n2.Anti-Clockwise Rotation ";
    cout << "\n\nEnter your choice(1-2): ";
    cin >> type;
    Ang = (Ang * 6.2832)/360;
    Sinang = sin(Ang);
    Cosang = cos(Ang);
          cout<<"Mark1";
    for(i=0;i<3;i++)
    for(j=0;j<3;j++)
        Rotation_matrix[i][j] = 0;
        cout<<"Mark2";
    Rotation_matrix[0][0] =
Rotation_matrix[1][1] = Cosang;
    Rotation_matrix[0][1] =
Rotation_matrix[1][0] = Sinang;
    Rotation_matrix[2][2] = 1;
    if(type == 1)
        Rotation_matrix[0][1] = -Sinang;
    else
```

```
        Rotation_matrix[1][0] = -Sinang;


         for(i1=0;i1<10;i1++)
    for(j1=0;j1<10;j1++)
        Rotation_result[i1][j1] = 0;
    for(i1=0;i1<r1;i1++)
    for(j1=0;j1<c2;j1++)
    for(k1=0;k1<c1;k1++)
        Rotation_result[i1][j1] =
Rotation_result[i1][j1]+(p[i1][k1] *
Rotation_matrix[k1][j1]);


    cout << "\n\nPolygon after Rotation : ";
         for(i=0;i<n;i++)
    {
        cout<<"\n";
        for(int j=0;j<3;j++)
        {

    cout<<Rotation_result[i][j]<<"\t\t";
        }
    }
    draw_polyfloat(Rotation_result,n);
}

void POLYGON :: scaling(int p[][10],int n)
```

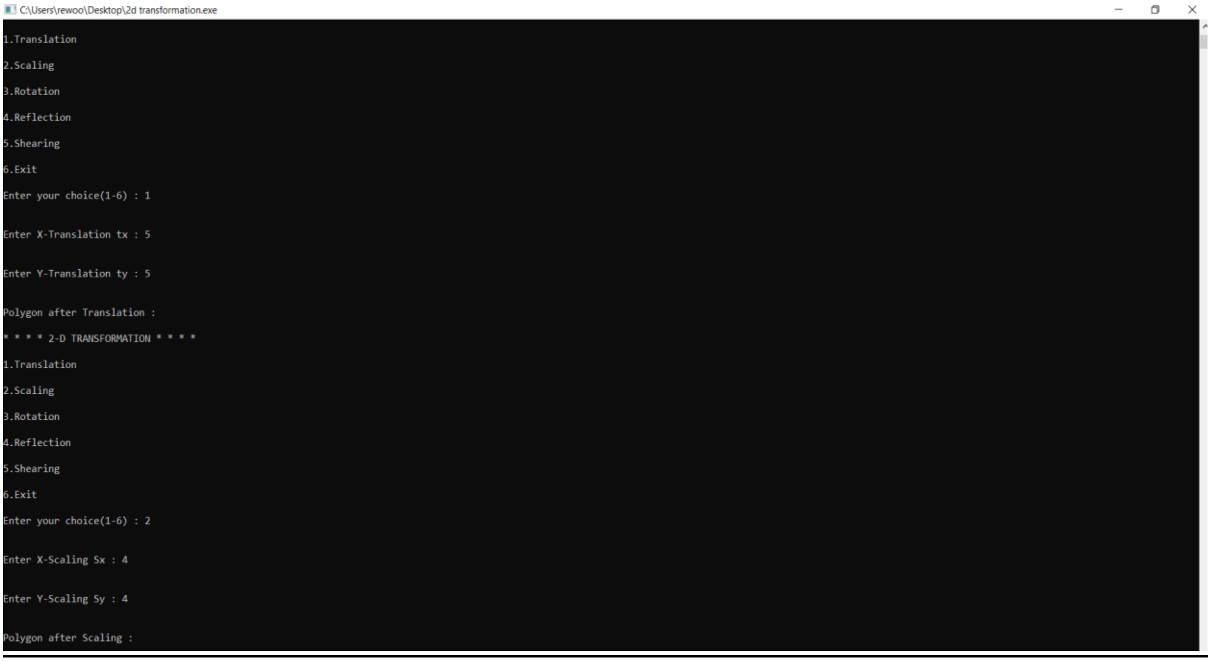
```
{
    float Sx,Sy;
         int i,j; int i1,j1,k1,r1,c1,c2;
         r1=n;c1=c2=3;
    cout<<"\n\nEnter X-Scaling Sx : ";
    cin>>Sx;
    cout<<"\n\nEnter Y-Scaling Sy : ";
    cin>>Sy;

    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            Scaling_matrix[i][j] = 0;
        }
    }

    Scaling_matrix[0][0] = Sx;
    Scaling_matrix[0][1] = 0;
    Scaling_matrix[0][2] = 0;
    Scaling_matrix[1][0] = 0;
    Scaling_matrix[1][1] = Sy;
    Scaling_matrix[1][2] = 0;
    Scaling_matrix[2][0] = 0;
    Scaling_matrix[2][1] = 0;
    Scaling_matrix[2][2] = 1;


        for(i1=0;i1<10;i1++)
    for(j1=0;j1<10;j1++)
        Scaling_result[i1][j1] = 0;
    for(i1=0;i1<r1;i1++)
    for(j1=0;j1<c2;j1++)
    for(k1=0;k1<c1;k1++)
        Scaling_result[i1][j1] =
Scaling_result[i1][j1]+(p[i1][k1] *
Scaling_matrix[k1][j1]);


    cout<<"\n\nPolygon after Scaling : ";
    draw_polyfloat(Scaling_result,n);
}


void POLYGON :: shearing(int p[][10],int n)
{
    float Sx,Sy,type; int i,j;
        int i1,j1,k1,r1,c1,c2;
        r1=n;c1=c2=3;
    for(i=0;i<3;i++)
    for(j=0;j<3;j++)
    {
        if(i == j)
            Shearing_matrix[i][j] = 1;
        else
            Shearing_matrix[i][j] = 0;
    }
    cout << "\n\n* * * * Shearing Types * * * 
*";
    cout << "\n\n1.X-Direction Shear \n\n2.Y-
Direction Shear ";
    cout << "\n\nEnter your choice(1-2) : ";
    cin >> type;
    if(type == 1)
    {
        cout << "\n\nEnter X-Shear Sx : ";
        cin >> Sx;
        Shearing_matrix[1][0] = Sx;
    }
    else
    {
        cout << "\n\nEnter Y-Shear Sy : ";
        cin >> Sy;
        Shearing_matrix[0][1] = Sy;
    }



         for(i1=0;i1<10;i1++)
    for(j1=0;j1<10;j1++)
    Shearing_result[i1][j1] = 0;
    for(i1=0;i1<r1;i1++)
    for(j1=0;j1<c2;j1++)
    for(k1=0;k1<c1;k1++)
        Shearing_result[i1][j1] =
Shearing_result[i1][j1]+(p[i1][k1] *
Shearing_matrix[k1][j1]);

    cout << "\n\nPolygon after Shearing : ";
    draw_polyfloat(Shearing_result,n);
}

void POLYGON :: reflection(int p[][10],int n)
{
    int type,i,j;

         int i1,j1,k1,r1,c1,c2;
  r1=n;c1=c2=3;
    cout << "\n\n* * * * Reflection Types * * *
*";
    cout << "\n\n1.About X-Axis \n\n2.About Y-
Axis \n\n3.About Origin\n\n4.About Line y = x
\n\n5.About Line y = -x \n\nEnter your
choice(1-5) : ";
    cin >> type;
    for(i=0;i<3;i++)
    for(j=0;j<3;j++)
    {
            Reflection_matrix[i][j] = 0;
    }
    switch(type)
    {
        case 1:
            Reflection_matrix[0][0] = 1;
                        Reflection_matrix[1][1]
= -1;
                        Reflection_matrix[2][2]
= 1;
            break;
        case 2:
            Reflection_matrix[0][0] = -1;
                        Reflection_matrix[1][1]
= 1;
                        Reflection_matrix[2][2]
= 1;
            break;
        case 3:
            Reflection_matrix[0][0] = -1;
                        Reflection_matrix[1][1]
= -1;
                        Reflection_matrix[2][2]
= 1;
            break;
        case 4:
            Reflection_matrix[0][1] = 1;
            Reflection_matrix[1][0] = 1;
            Reflection_matrix[2][2] = 1;
            break;
        case 5:
            Reflection_matrix[0][1] = -1;
            Reflection_matrix[1][0] = -1;
            Reflection_matrix[2][2] = 1;
            break;
    }


        for(i1=0;i1<10;i1++)
    for(j1=0;j1<10;j1++)
        Reflection_result[i1][j1] = 0;
    for(i1=0;i1<r1;i1++)
    for(j1=0;j1<c2;j1++)
    for(k1=0;k1<c1;k1++)
        Reflection_result[i1][j1] =
Reflection_result[i1][j1]+(p[i1][k1] *
Reflection_matrix[k1][j1]);


    cout << "\n\n\t\tPolygon after Reflection :
";
//cout << "\n\n\t\tPolygon after Rotationâ€¦";
        for(i=0;i<n;i++)
    {
        cout<<"\n";
        for(int j=0;j<3;j++)
        {

    cout<<Reflection_result[i][j]<<"\t\t";
        }
    }
    draw_poly(Reflection_result,n);
//closegraph();
}




int main()
{
    int ch,n,p[10][10];
    POLYGON p1;
    cout<<"\n\n* * * * 2-D TRANSFORMATION * * *
*";
    n= p1.accept_poly(p);

    cout <<"\n\nOriginal Polygon : ";
    p1.draw_poly(p,n);
    do
    {

                int ch;
```

```
            cout<<"\n\n* * * * 2-D
TRANSFORMATION * * * *";
        cout<<"\n\n1.Translation \n\n2.Scaling
\n\n3.Rotation \
            \n\n4.Reflection \n\n5.Shearing
\n\n6.Exit";
            cout<<"\n\nEnter your choice(1-6) :
";
            cin>>ch;

        switch(ch)
        {
            case 1:
                p1.translation(p,n);
                break;

            case 2:
                p1.scaling(p,n);
                break;

            case 3:
                p1.rotation(p,n);
                break;

            case 4:
                p1.reflection(p,n);
                break;

            case 5:
                p1.shearing(p,n);
                break;

            case 6:
                exit(0);
        }
    }while(1);
    return 0;
}
```

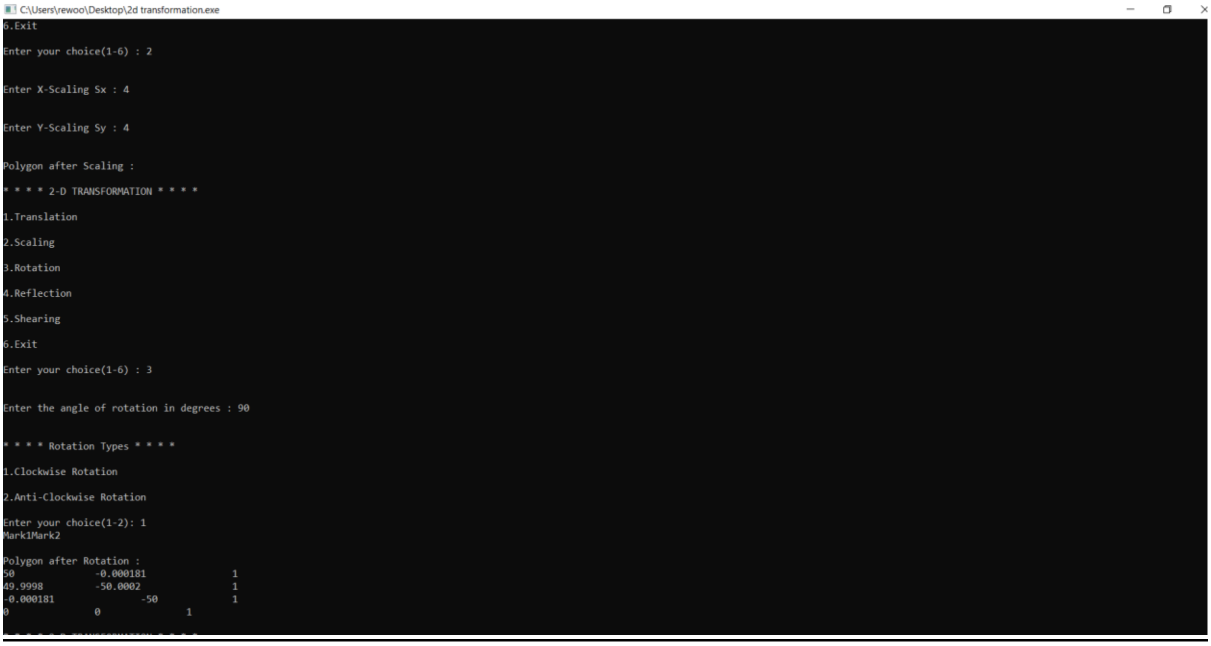
OUTPUT:



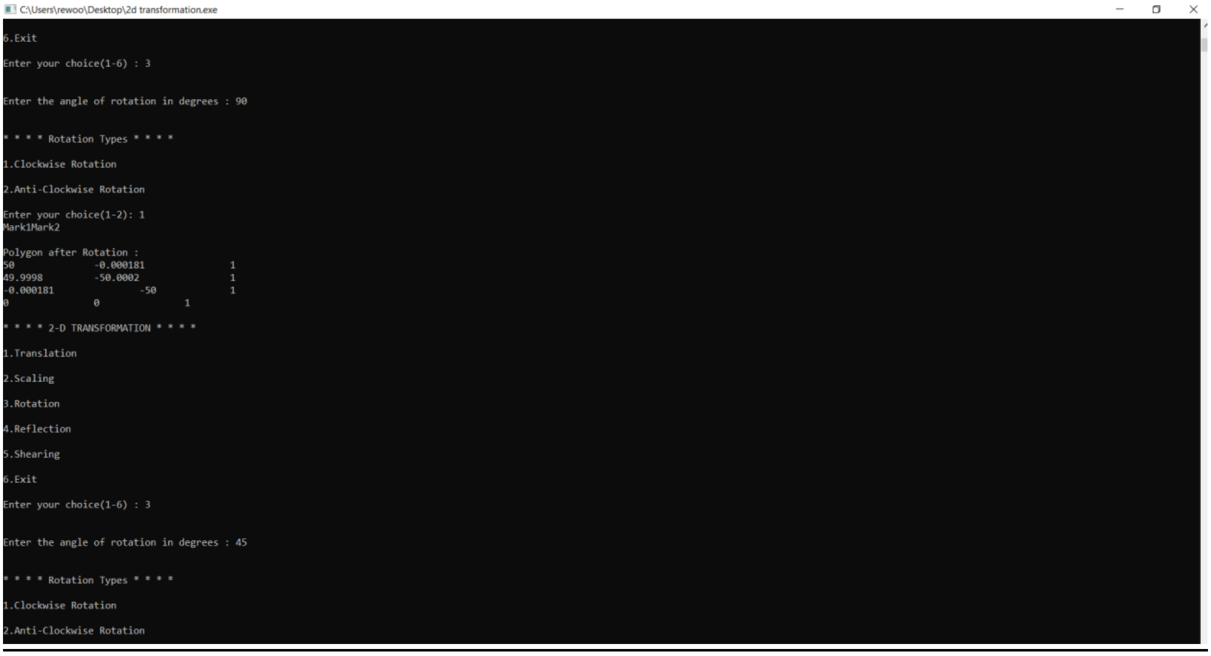
```
* * * * 2-D TRANSFORMATION * * * *

Enter number of vertices : 4


Enter (x,y) Co-ordinate of point P0 : 0
50


Enter (x,y) Co-ordinate of point P1 : 50
50


Enter (x,y) Co-ordinate of point P2 : 50
0


Enter (x,y) Co-ordinate of point P3 : 0
0

0               50              1
50              50              1
50              0               1
0               0               1

Original Polygon :

* * * * 2-D TRANSFORMATION * * * *

1.Translation

2.Scaling

3.Rotation

4.Reflection

5.Shearing

6.Exit

Enter your choice(1-6) : 1


Enter X-Translation tx : 5
```



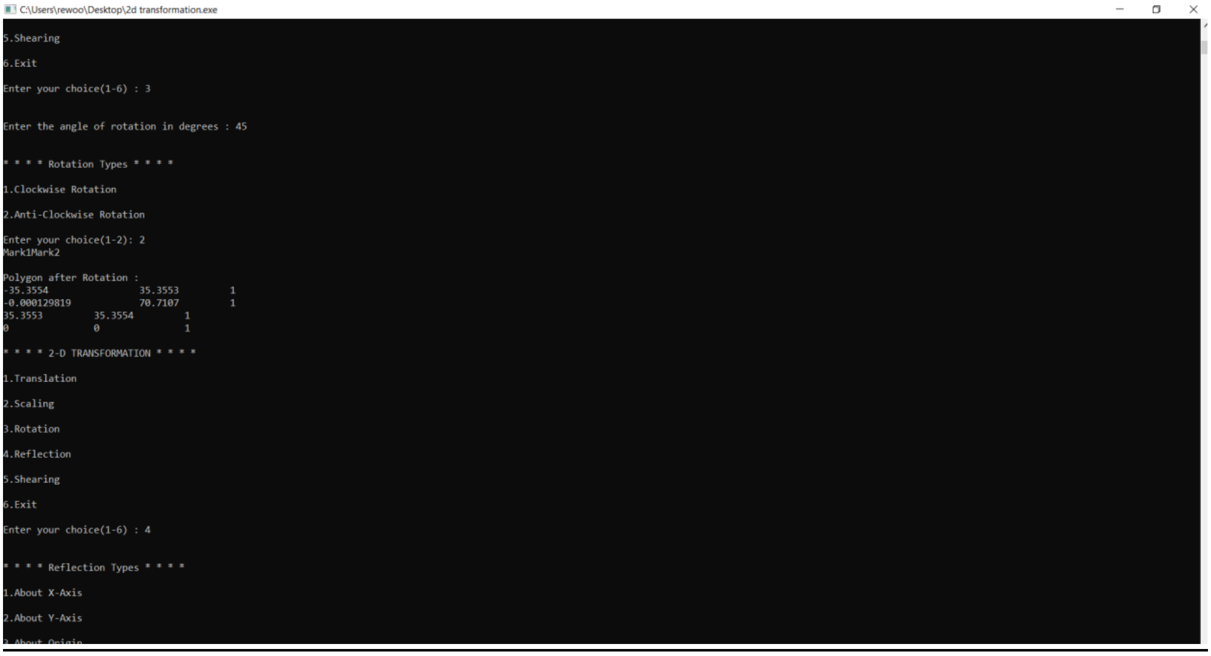
```
1.Translation

2.Scaling

3.Rotation

4.Reflection

5.Shearing

6.Exit

Enter your choice(1-6) : 1


Enter X-Translation tx : 5


Enter Y-Translation ty : 5


Polygon after Translation :

* * * * 2-D TRANSFORMATION * * * *

1.Translation

2.Scaling

3.Rotation

4.Reflection

5.Shearing

6.Exit

Enter your choice(1-6) : 2


Enter X-Scaling Sx : 4


Enter Y-Scaling Sy : 4


Polygon after Scaling :
```



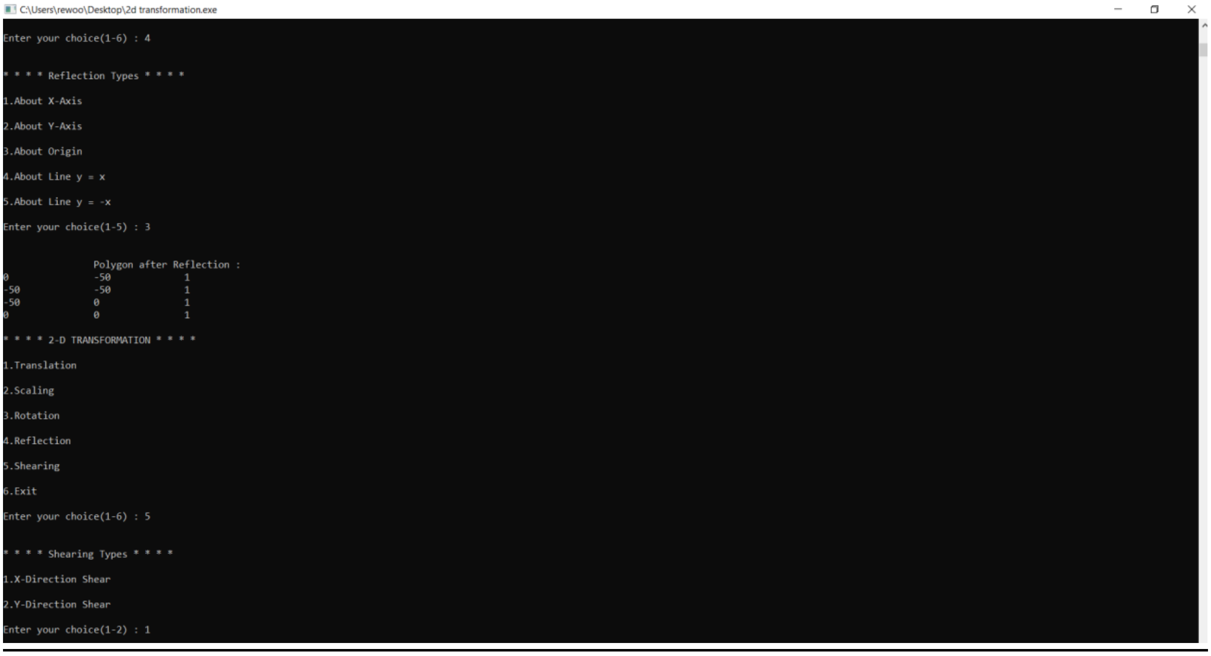
```
6.Exit

Enter your choice(1-6) : 2


Enter X-Scaling Sx : 4


Enter Y-Scaling Sy : 4


Polygon after Scaling :

* * * * 2-D TRANSFORMATION * * * *

1.Translation

2.Scaling

3.Rotation

4.Reflection

5.Shearing

6.Exit

Enter your choice(1-6) : 3


Enter the angle of rotation in degrees : 90


* * * * Rotation Types * * * *

1.Clockwise Rotation

2.Anti-Clockwise Rotation

Enter your choice(1-2): 1
Mark1Mark2

Polygon after Rotation :
50              -0.000181               1
49.9998         -50.0002                1
-0.000181               -50             1
0               0               1
```



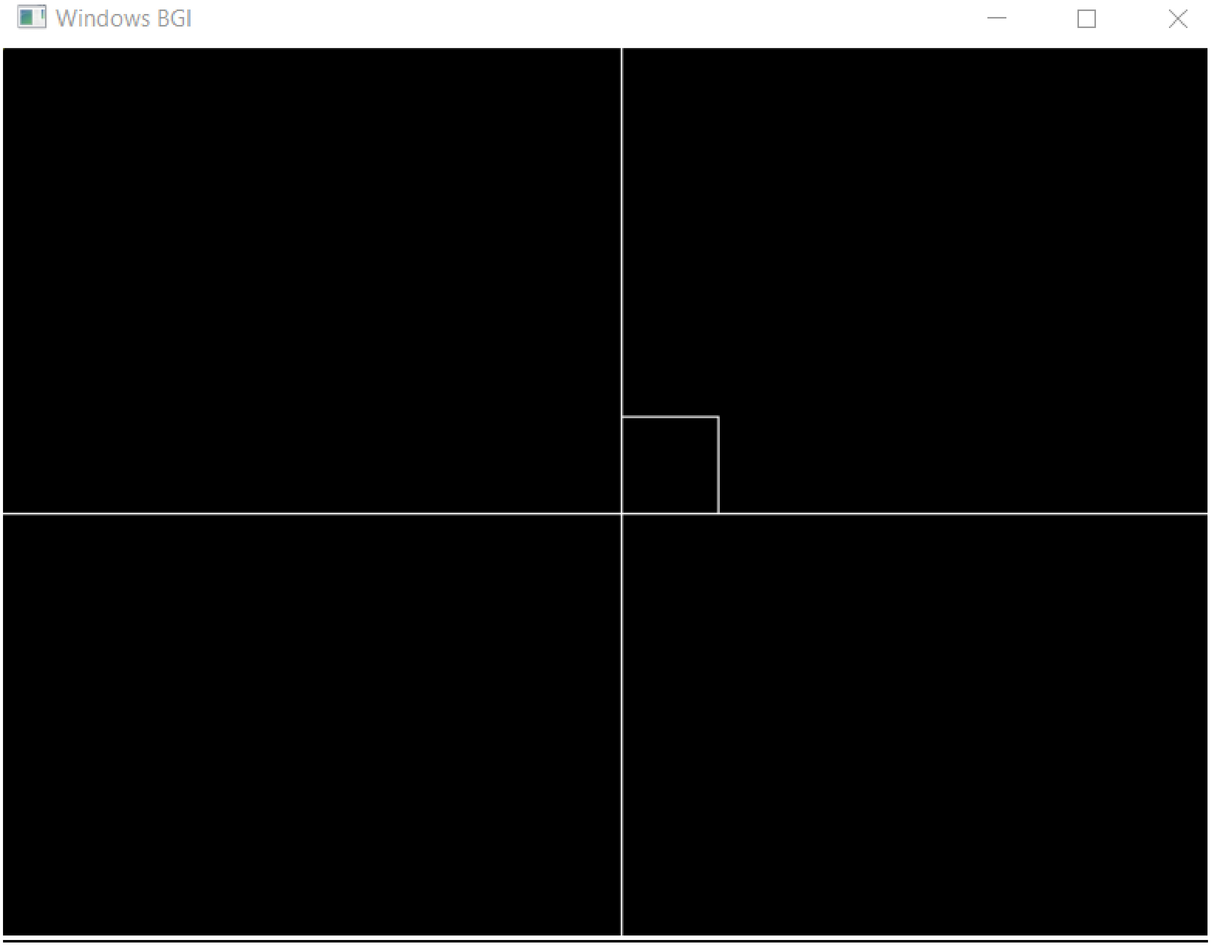
```
6.Exit

Enter your choice(1-6) : 3


Enter the angle of rotation in degrees : 90


* * * * Rotation Types * * * *

1.Clockwise Rotation

2.Anti-Clockwise Rotation

Enter your choice(1-2): 1
Mark1Mark2

Polygon after Rotation :
50              -0.000181               1
49.9998         -50.0002                1
-0.000181               -50             1
0               0               1

* * * * 2-D TRANSFORMATION * * * *

1.Translation

2.Scaling

3.Rotation

4.Reflection

5.Shearing

6.Exit

Enter your choice(1-6) : 3


Enter the angle of rotation in degrees : 45


* * * * Rotation Types * * * *

1.Clockwise Rotation

2.Anti-Clockwise Rotation
```

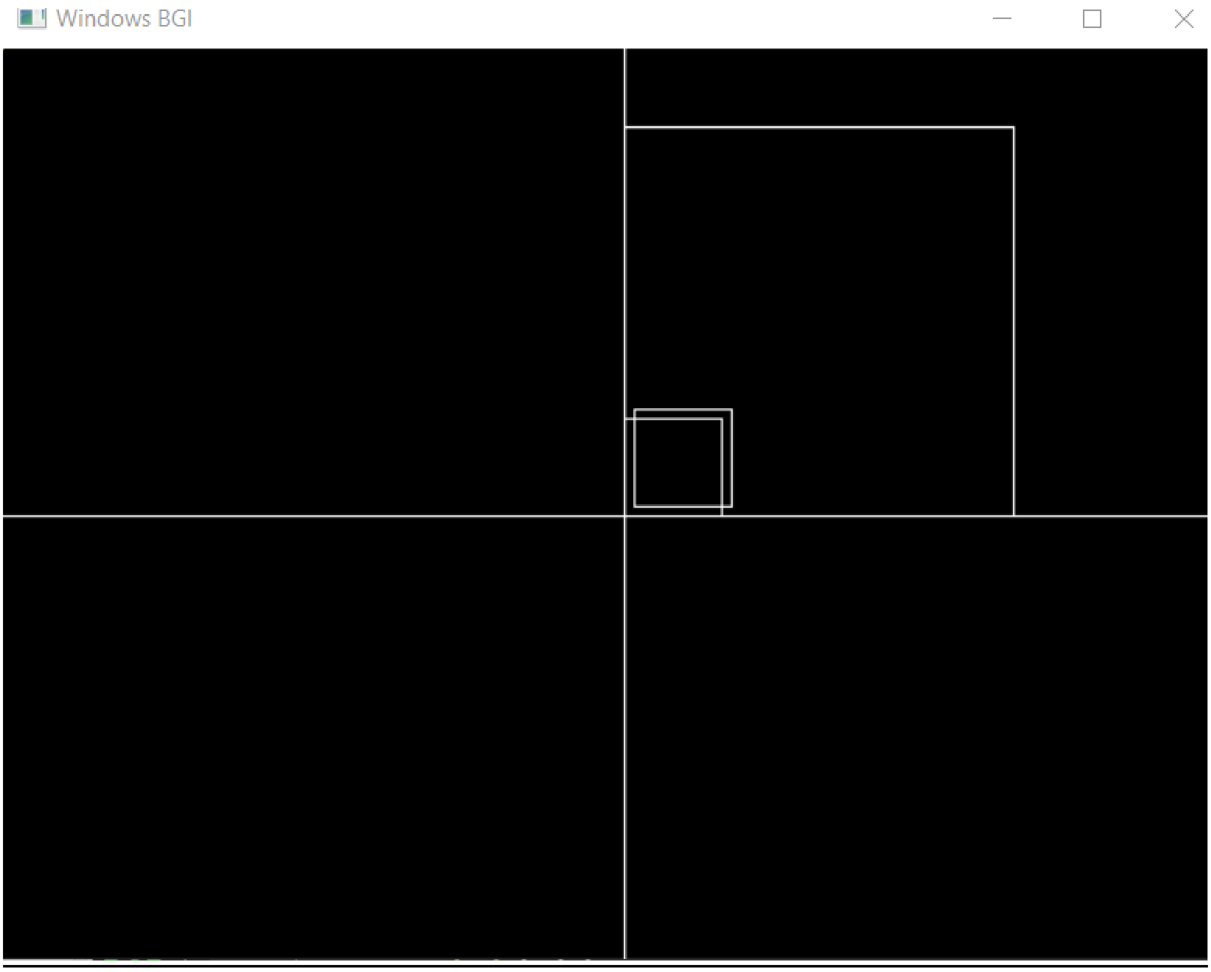


```
5.Shearing

6.Exit

Enter your choice(1-6) : 3


Enter the angle of rotation in degrees : 45


* * * * Rotation Types * * * *

1.Clockwise Rotation

2.Anti-Clockwise Rotation

Enter your choice(1-2): 2
Mark1Mark2

Polygon after Rotation :
-35.3554                35.3553         1
-0.000129819            70.7107         1
35.3553         35.3554         1
0               0               1

* * * * 2-D TRANSFORMATION * * * *

1.Translation

2.Scaling

3.Rotation

4.Reflection

5.Shearing

6.Exit

Enter your choice(1-6) : 4


* * * * Reflection Types * * * *

1.About X-Axis

2.About Y-Axis

3.About Origin
```

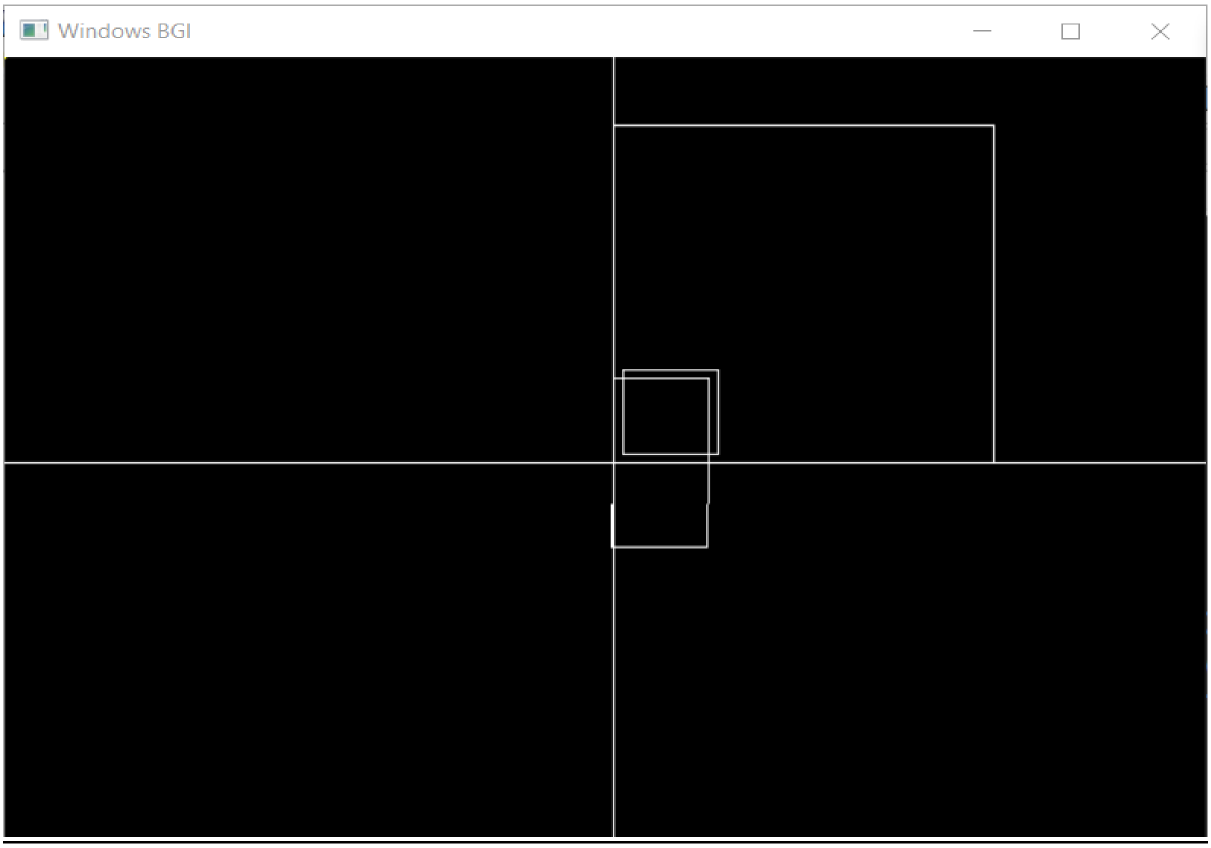


```
Enter your choice(1-6) : 4

* * * * Reflection Types * * * *

1.About X-Axis

2.About Y-Axis

3.About Origin

4.About Line y = x

5.About Line y = -x

Enter your choice(1-5) : 3


                Polygon after Reflection :
0               -50             1
-50             -50             1
-50             0               1
0               0               1

* * * * 2-D TRANSFORMATION * * * *

1.Translation

2.Scaling

3.Rotation

4.Reflection

5.Shearing

6.Exit

Enter your choice(1-6) : 5

* * * * Shearing Types * * * *

1.X-Direction Shear

2.Y-Direction Shear

Enter your choice(1-2) : 1
```

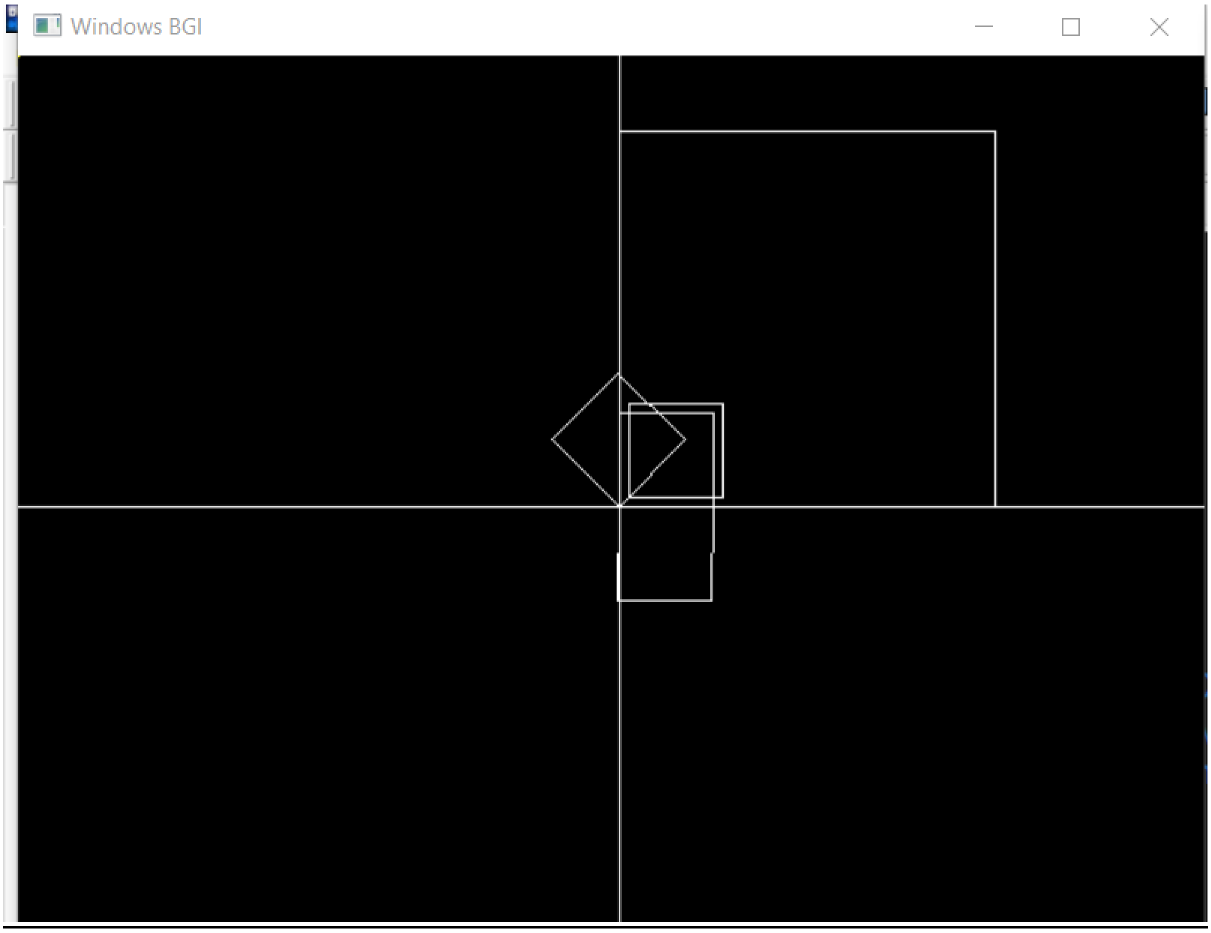




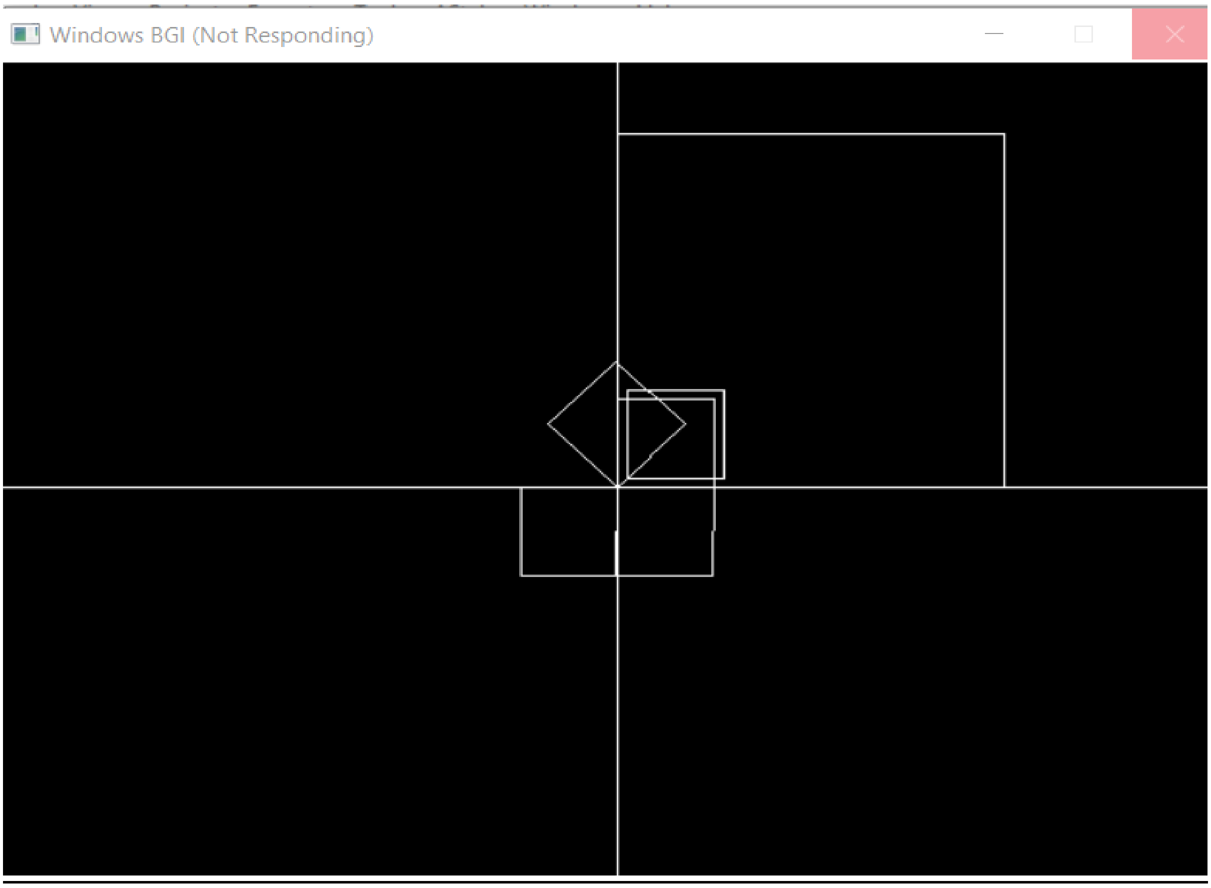
Windows BGI



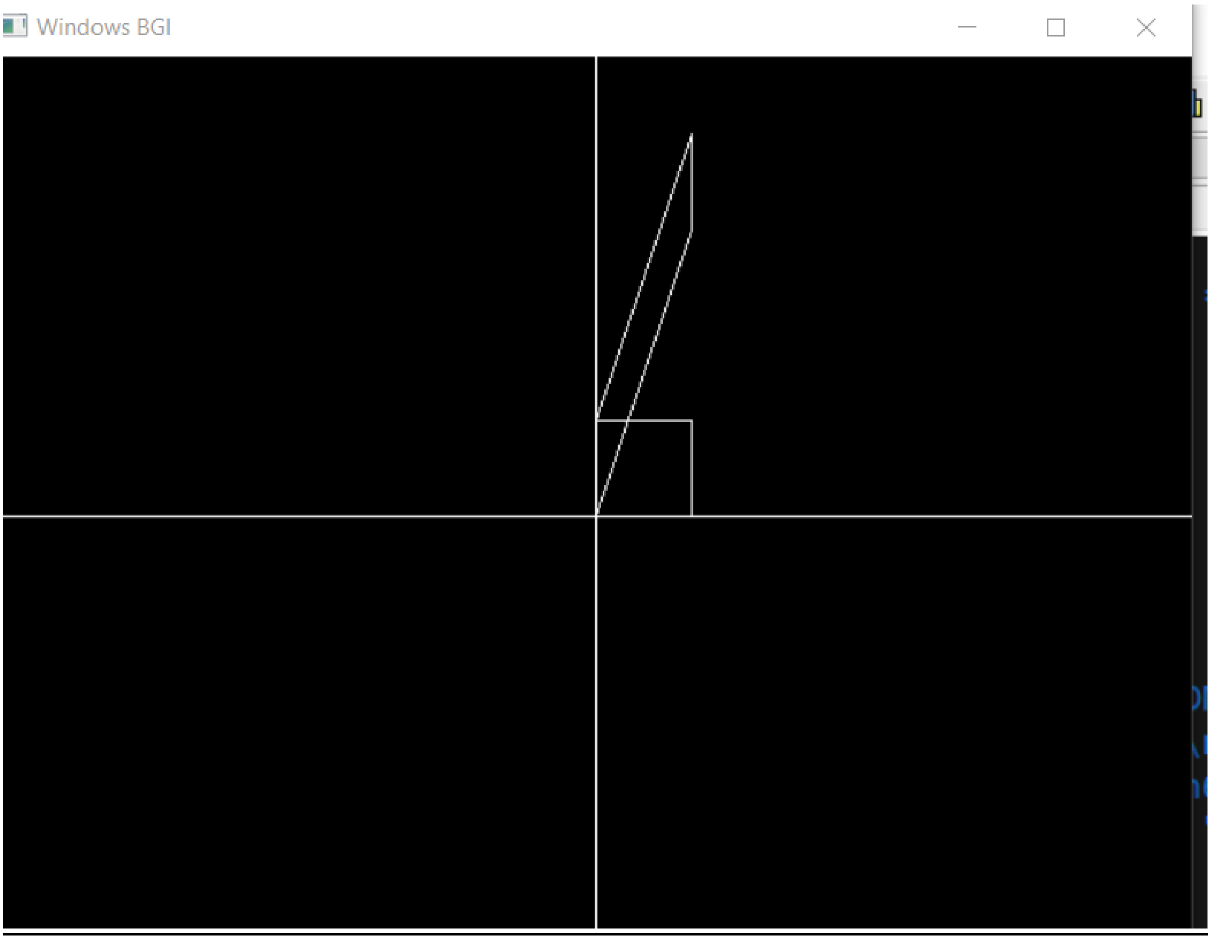
Windows BGI

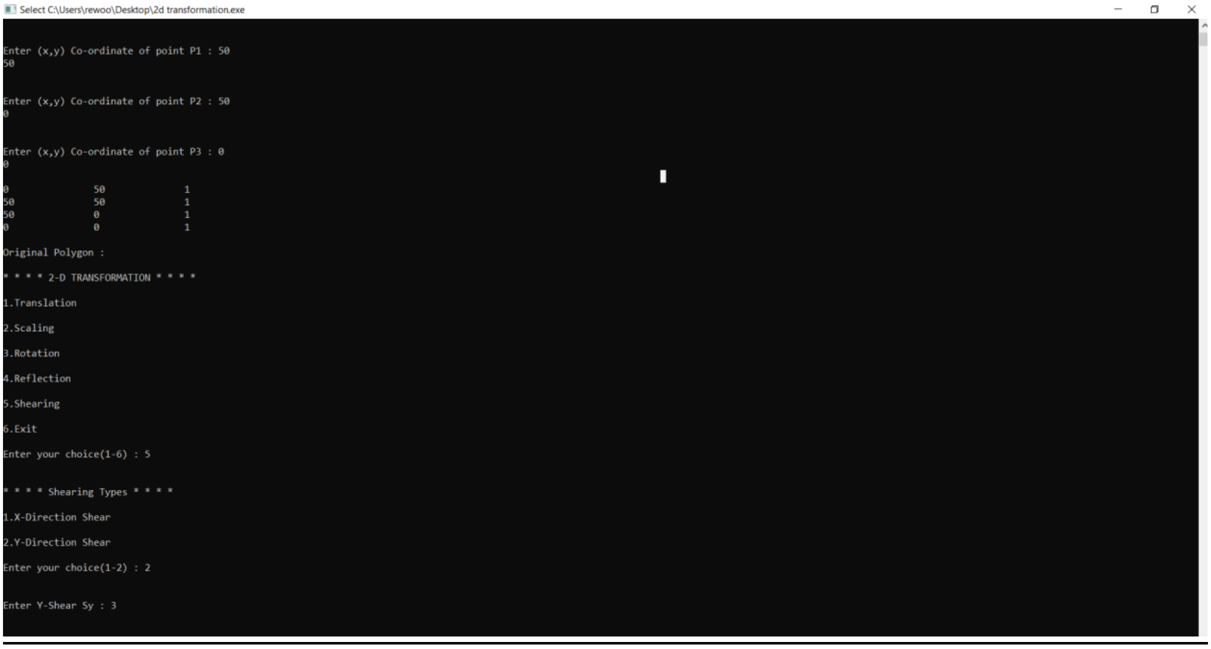


Windows BGI



Windows BGI (Not Responding)





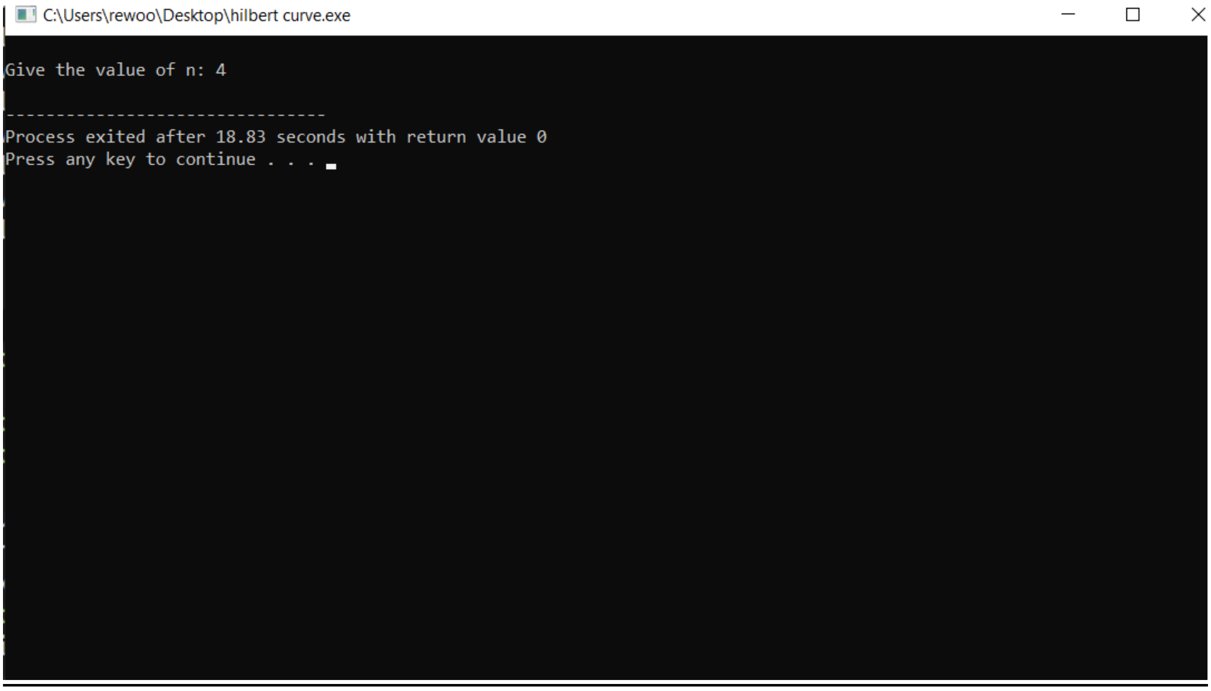
```
Enter (x,y) Co-ordinate of point P1 : 50
50


Enter (x,y) Co-ordinate of point P2 : 50
0


Enter (x,y) Co-ordinate of point P3 : 0
0

0               50              1
50              50              1
50              0               1
0               0               1

Original Polygon :

* * * * 2-D TRANSFORMATION * * * *

1.Translation

2.Scaling

3.Rotation

4.Reflection

5.Shearing

6.Exit

Enter your choice(1-6) : 5


* * * * Shearing Types * * * *

1.X-Direction Shear

2.Y-Direction Shear

Enter your choice(1-2) : 2


Enter Y-Shear Sy : 3
```

Practical 5

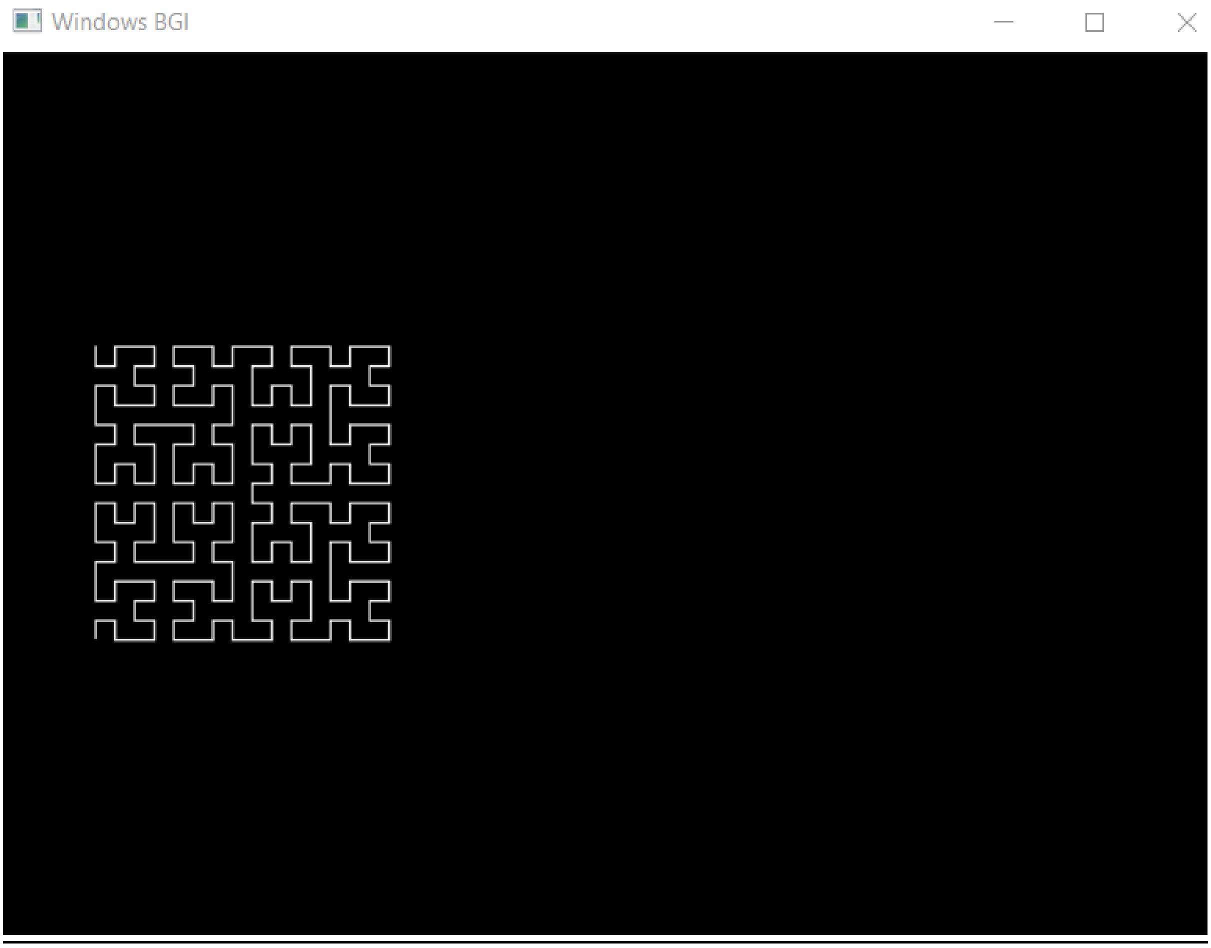
```
/* Write C++ Program to generate Hilbert Curve
using concept of fractals. */

#include <iostream>
#include <stdlib.h>
#include <graphics.h>
#include <math.h>

using namespace std;

void move(int j,int h,int &x,int &y)
{
if(j==1)
y-=h;
else if(j==2)
x+=h;
else if(j==3)
y+=h;
else if(j==4)
x-=h;
lineto(x,y);
}

void hilbert(int r,int d,int l,int u,int i,int
h,int &x,int &y)
```

```
{
if(i>0)
{
i--;
hilbert(d,r,u,l,i,h,x,y);
move(r,h,x,y);
hilbert(r,d,l,u,i,h,x,y);
move(d,h,x,y);
hilbert(r,d,l,u,i,h,x,y);
move(l,h,x,y);
hilbert(u,l,d,r,i,h,x,y);
}
}

int main()
{
int n,x1,y1;
int x0=50,y0=150,x,y,h=10,r=2,d=3,l=4,u=1;

cout<<"\nGive the value of n: ";
cin>>n;
x=x0;y=y0;
int gm,gd=DETECT;
initgraph(&gd,&gm,NULL);
moveto(x,y);
hilbert(r,d,l,u,n,h,x,y);
delay(10000);

closegraph();

return 0;
}
```

OUTPUT:



```
Give the value of n: 4

--------------------------------
Process exited after 18.83 seconds with return value 0
Press any key to continue . . .
```



Windows BGI

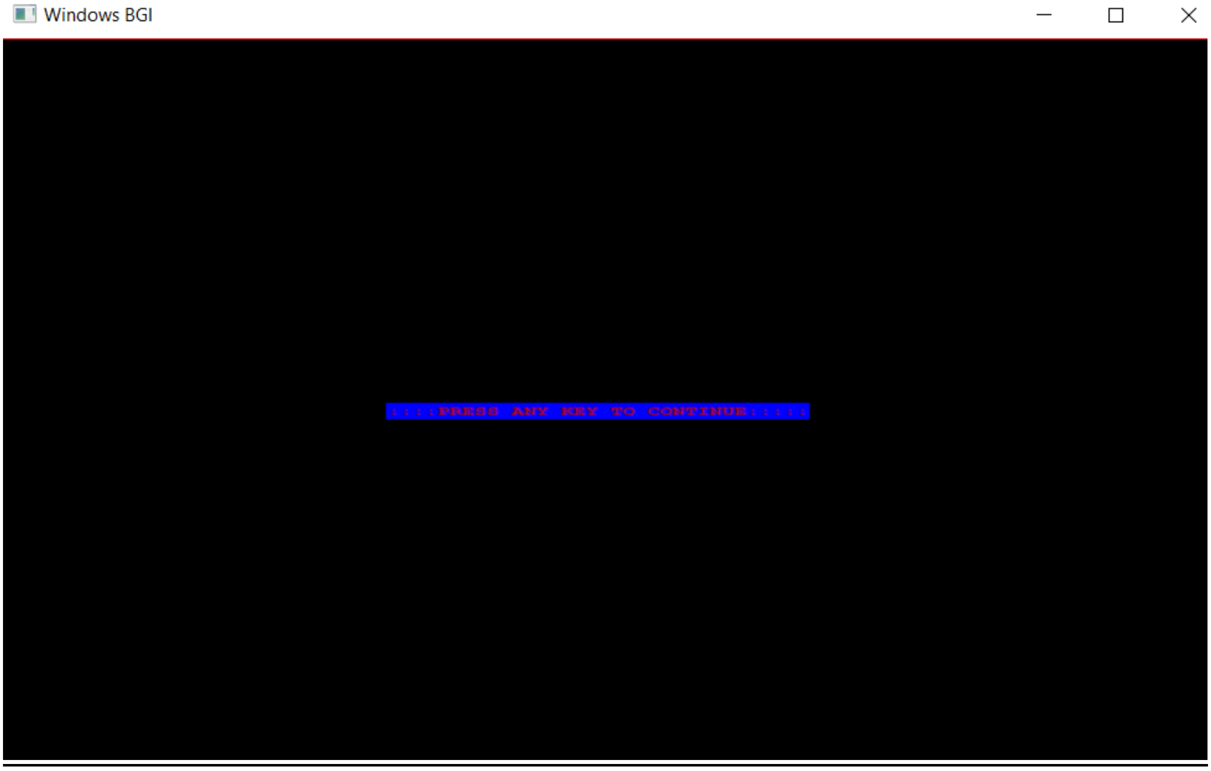
Practical 6:

```
/* Write a program to draw Sunrise and Sunset.
*/


#include<iostream>

#include<graphics.h>

#include<cstdlib>

#include<dos.h>

#include<cmath>

using namespace std;


int main()

{

    initwindow(800,500);

    int x0,y0;

    int gdriver = DETECT,gmode,errorcode;

    int xmax,ymax;


    errorcode=graphresult();


    if(errorcode!=0)

    {

        cout<<"Graphics
error:"<<grapherrormsg(errorcode);

        cout<<"Press any ket to halt";

        exit(1);
    }
    int i,j;
    setbkcolor(BLUE);
    setcolor(RED);
    rectangle(0,0,getmaxx(),getmaxy());

    outtextxy(250,240,"::::PRESS ANY KEY TO
CONTINUE:::::");
    while(!kbhit());
    for(i=50,j=0;i<=250,j<=250;i+=5,j+=5)
    {
        delay(120);
        cleardevice();
        if(i<=150)
        {
            setcolor(YELLOW);
            setfillstyle(1,YELLOW);
            fillellipse(i,300-j,20,20);
        }
        else
        {
            setcolor(GREEN^RED);
            setfillstyle(1,GREEN^RED);
            fillellipse(i,300-j,20,20);
        }
```

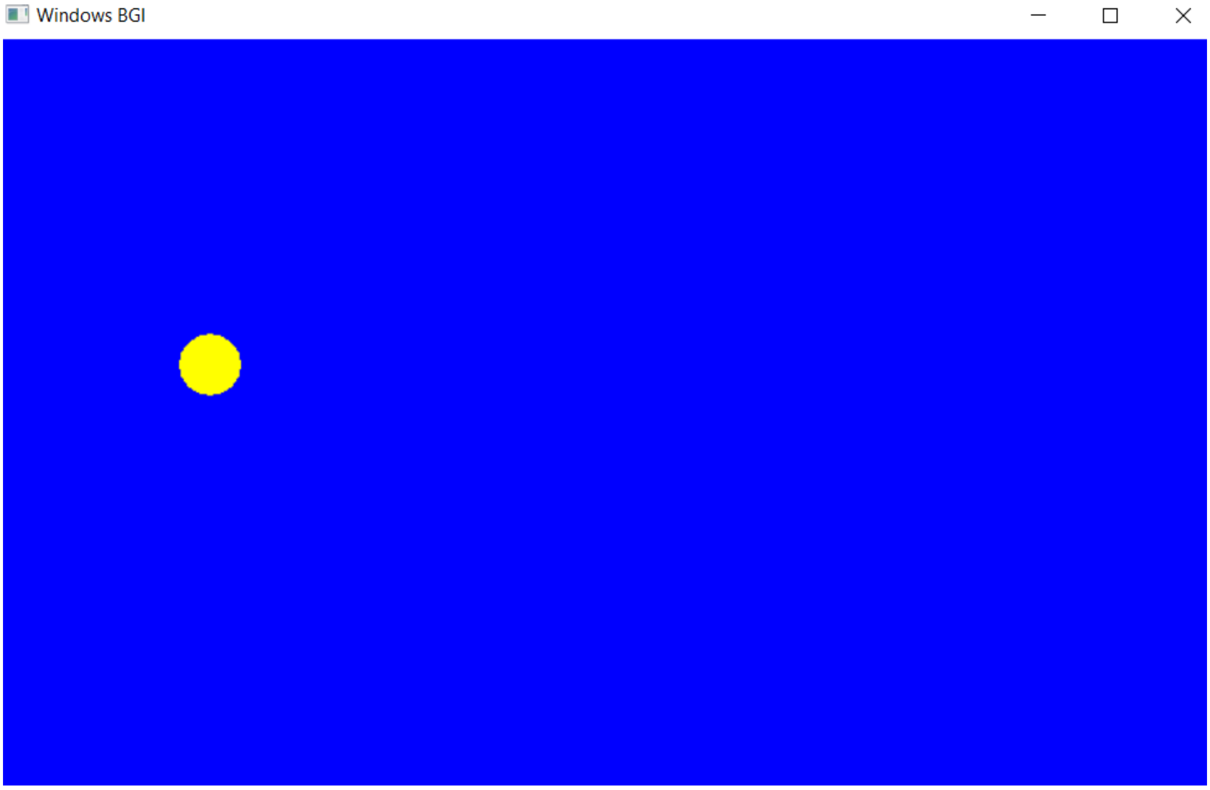
```
}
delay(1000);
cleardevice();
setcolor(RED);
setfillstyle(1,RED);
fillellipse(300,50,20,20);
delay(150);

int k,l;
for(k=305,l=55;k<=550,l<=300;k+=5,l+=5)
{
    delay(120);
    cleardevice();
    if(k<=450)
    {

        setcolor(GREEN^RED);
        setfillstyle(1,GREEN^RED);

        fillellipse(k,l,20,20);
    }
    else
    {
        setcolor(YELLOW);
        setfillstyle(1,YELLOW);
```

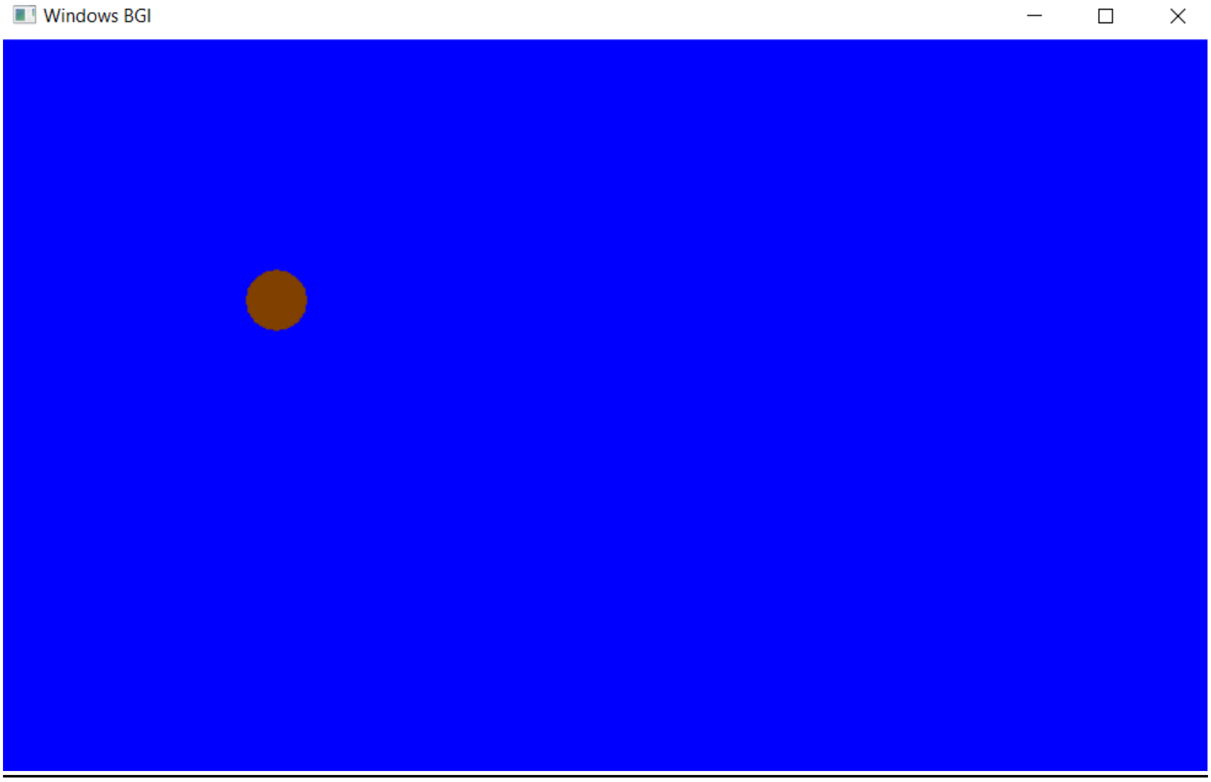
```
                fillellipse(k,l,20,20);
            }
        }
        return 0;
    }
```

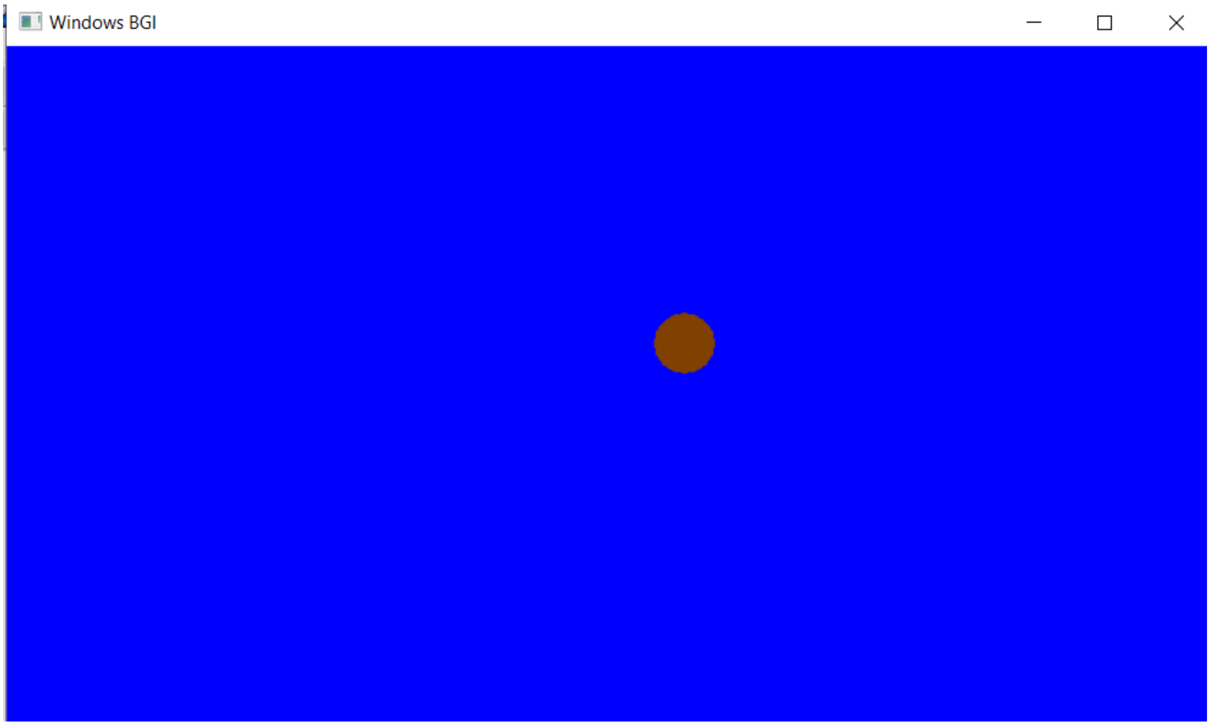
OUTPUT:

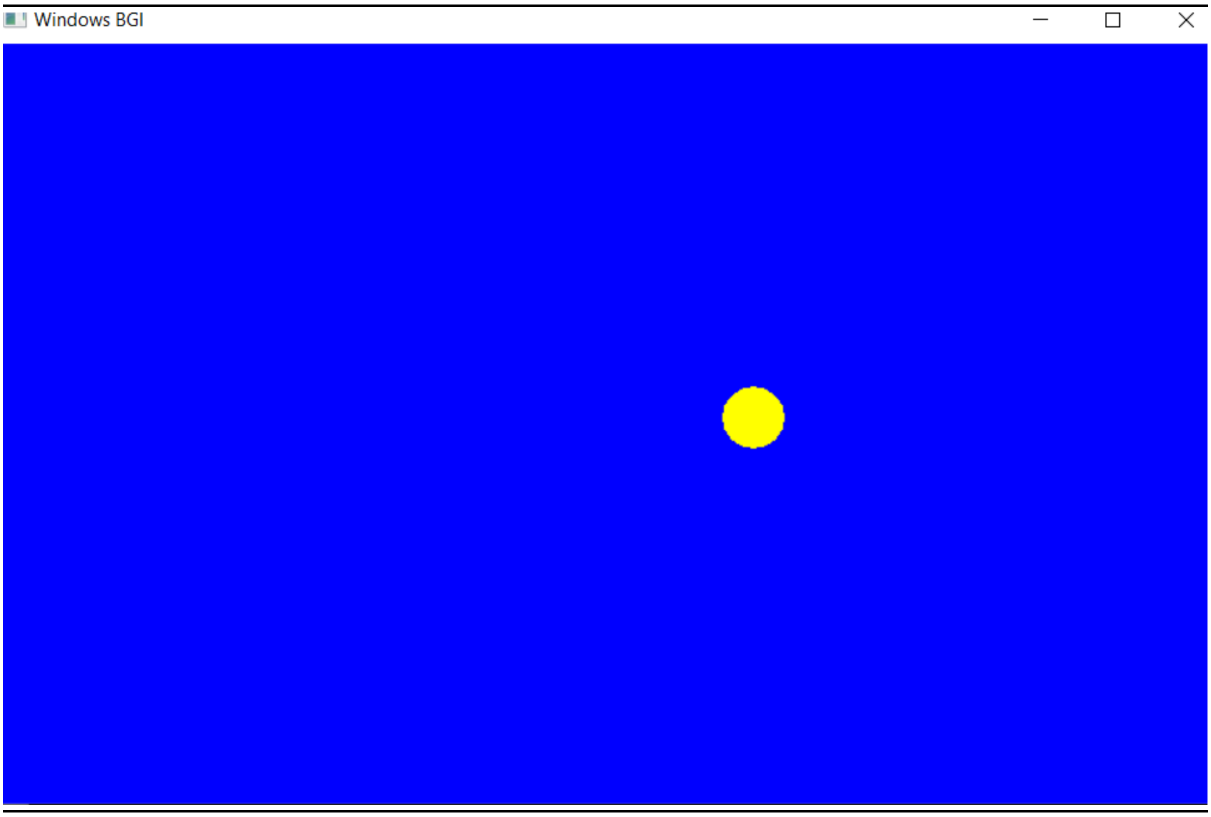


```
::::PRESS ANY KEY TO CONTINUE:::::
```


Windows BGI


Windows BGI


Windows BGI


Windows BGI