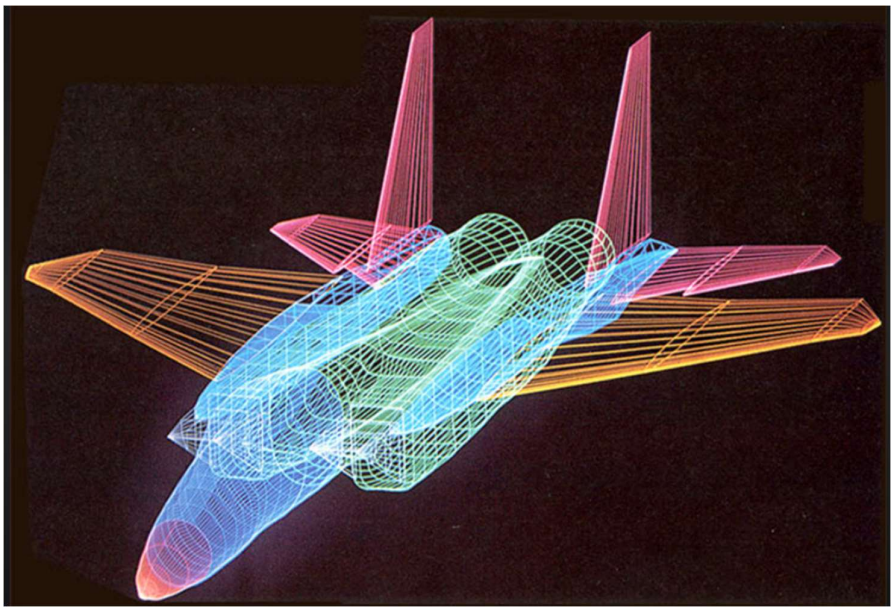
COMPUTER GRAPHICS

PRACTICAL FILE



Submitted By:

Name: Lakshay Sharma

College Roll No: 19/78013

Exam roll no.-19003570007

Course: BSc(H) Computer Science

Sem: VI

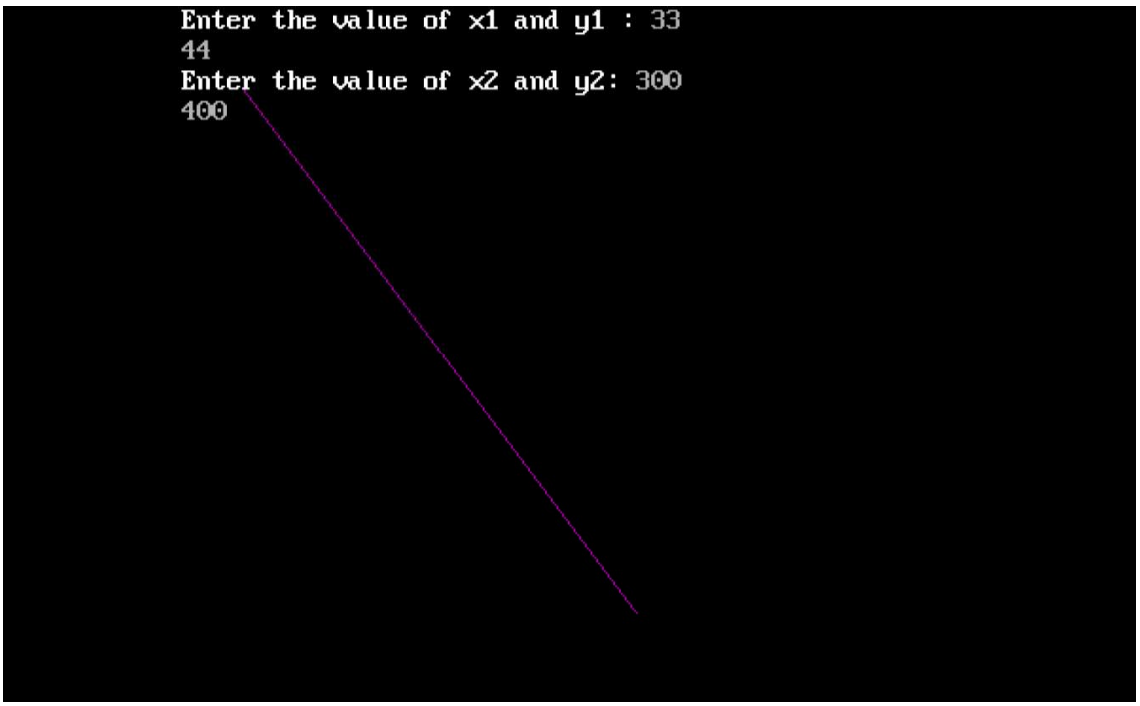
1. DDA Line Drawing Algorithm

Code:

```
//DDA line Drawing Algorithm
#include <graphics.h>
#include <iostream.h>
#include <math.h>
#include <dos.h>
#include <conio.h>
void main( )
{
clrscr();
float x,y,x1,y1,x2,y2,dx,dy,slope;
int gdriver = DETECT,gmode;
int i;
        initgraph(&gdriver,&gmode,"C:\\TC\\BGI");

cout<<"Enter the value of x1 and y1 : ";
cin>>x1>>y1;
cout<<"Enter the value of x2 and y2: ";
cin>>x2>>y2;

dx=abs(x2-x1);
dy=abs(y2-y1);
```

```
if(dx>=dy)// m<=1
slope=dx;
else//m>1
slope=dy;

dx=dx/slope;
dy=dy/slope;

x=x1;
y=y1;

i=1;
while(i<=slope)
{
putpixel(x,y,5);
x=x+dx;
y=y+dy;
i=i+1;
delay(50);
}
closegraph();
getch();
}
```

OUTPUT-


Enter the value of x1 and y1 : 33
44
Enter the value of x2 and y2: 300
400

2. BRESENHAM'S Line Drawing Algorithm

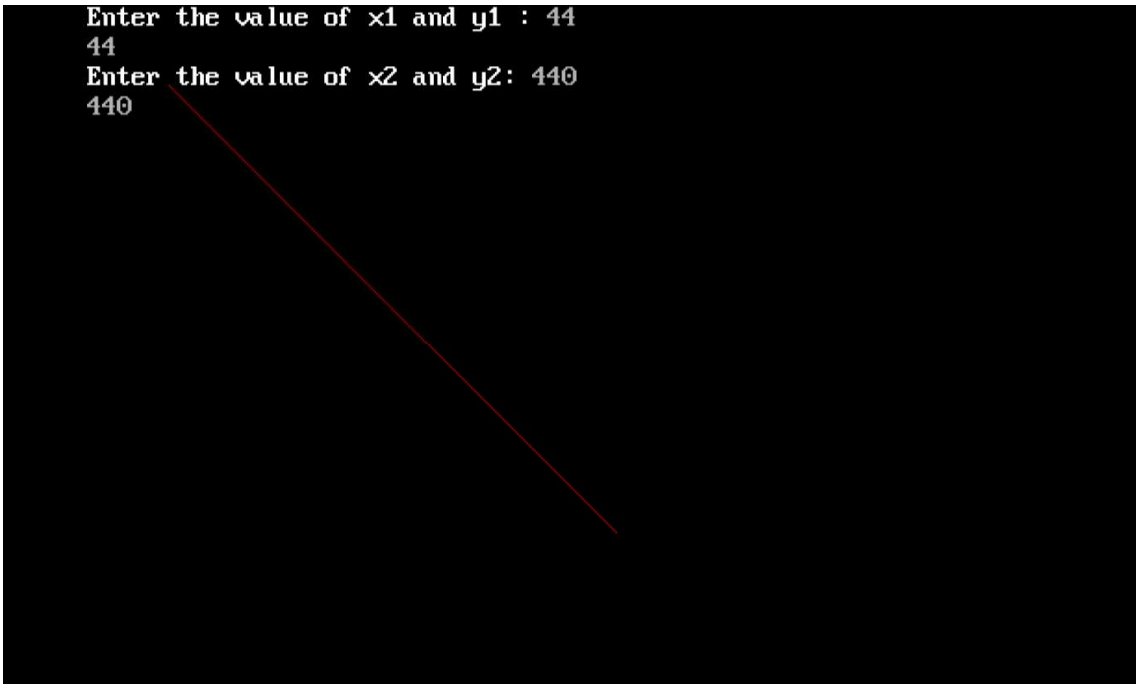
Code:

```
//Bresenham's line drawing algorithm
#include <graphics.h>
#include <iostream.h>
#include <math.h>
#include <dos.h>
#include <conio.h>
void main( )
{
clrscr();
float x,y,x1,y1,x2,y2,dx,dy,slope;
int gdriver = DETECT,gmode;
int i,d;
initgraph(&gdriver,&gmode,"C:\\TC\\BGI");

cout<<"Enter the value of x1 and y1 : ";
cin>>x1>>y1;
cout<<"Enter the value of x2 and y2: ";
cin>>x2>>y2;

dx=abs(x2-x1);
dy=abs(y2-y1);
```

```
int a,b;
a=2*dy;
b= -2*dx;
x=x1; y=y1;
int d1;
d=2*dy-dx;
while(x!=x2+1&&y!=y2+1)
{
 if(d>0)
 {
   putpixel(x,y,RED);
   d1=d+a+b;
   x++; y++;
   d=d1;
   delay(150);
 }
 else
 {
   putpixel(x,y,RED);
   d1=d+a;
   x++;
   d=d1;
   delay(100);
 }
}
```

```
closegraph();
}
```

Output:



3. BRESENHAM'S Circle Drawing Algorithm.

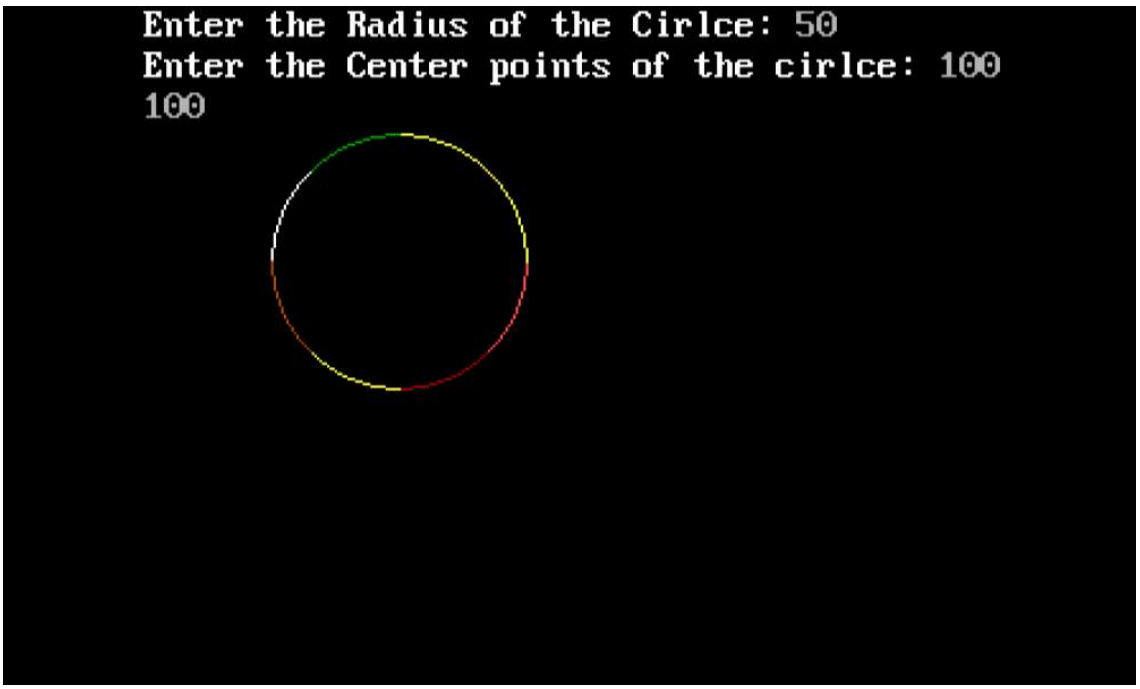
Code:

```
//Bresenham's Circle drawing algorithm
#include <graphics.h>
#include <iostream.h>
#include <math.h>
#include <dos.h>
#include <conio.h>
void display(int a,int b)
{
cout<<a<<", "<<b<<endl;
}
void drawCircle(int x,int y,int xc,int yc)
{
   putpixel(x+xc,y+yc,RED);
   putpixel(x+xc,-y+yc,YELLOW);
   putpixel(-x+xc,-y+yc,GREEN);
   putpixel(-x+xc,y+yc,YELLOW);
   putpixel(y+xc,x+yc,12);
   putpixel(y+xc,-x+yc,14);
   putpixel(-y+xc,-x+yc,15);
   putpixel(-y+xc,x+yc,6);
}
void main( )
```

```
{
clrscr();
int i,rad, d1;
float x,y,d,x1,y1;
int gdriver = DETECT,gmode;
initgraph(&gdriver,&gmode,"C:\\TC\\BGI");

cout<<"Enter the Radius of the Cirlce: ";
cin>>rad;
cout<<"Enter the Center points of the cirlce: ";
cin>>x1>>y1;
x=0;
y=rad;
d=5/4-rad;
while(x<=y)
{
 if(d<0)
 {
   drawCircle(x,y,x1,y1);
  // display(x,y);
   d1=d+4*x+6;
   x++;
   d=d1;
   delay(100);
 }
```

```
 else
 {
   drawCircle(x,y,x1,y1);
  // display(x,y);
   d1=d+4*x-4*y+10;
   x++; y--;
   d=d1;
   delay(100);
 }
}
getch();
closegraph();
}
```

Output:



4. Write a program to clip a line using Cohen and Sutherland line clipping algorithm.

Code:

```
#include <graphics.h>
#include <iostream.h>
#include <math.h>
#include <dos.h>
#include <conio.h>
int flag[4],flag1[4];
int x1,x2,y1,y2,x_min,y_min,x_max,y_max;
void display()
{
        rectangle(x_min,y_min,x_max,y_max);
        delay(100);
        line(x1,y1,x2,y2);

}
void point_verify1()//TBRL
{
        if(y1>y_max)
        {
        flag[0]=1;
```

```
        }
        if(y1<y_min)
        {
        flag[1]=1;
        }
        if(x1>x_max)
        {
        flag[2]=1;
        }
        if(x1<x_min)
        {
        flag[3]=1;
        }
}
void point_verify2() //TBRL
{
        if(y2>y_max)
        {
        flag1[0]=1;
        }
        if(y2<y_min)
        {
        flag1[1]=1;
        }
        if(x2>x_max)
```

```
        {
        flag1[2]=1;
        }
        if(x2<x_min)
        {
        flag1[3]=1;
        }
}
void verify()
{
        int a=0,b=0;
        for(int i=0;i<4;i++)
        {
                if(flag[i]!=0&&flag1[i]!=0)
                {
                a++;
                }
                else if(flag[i]==0&&flag1[i]==0)
                {
                b++;
                }
                else
                {
                b=0;
                }
```

```
}
if(a!=0)
{
	cout<<"The line cannot be clipped\n";
}
else if(b==0)
{
	cout<<"The line is partially inside the window\n";
	delay(1500);
	cleardevice();
	cout<<"Clipped line to the window is....\n";
	//Vertex 1
	if(x1<x_min)
	{
		x1=x_min;
	}
	if(x1>x_max)
	{
		x1=x_max;
	}
	if(y1<y_min)
	{
		y1=y_min;
	}
	if(y1>y_max)
```

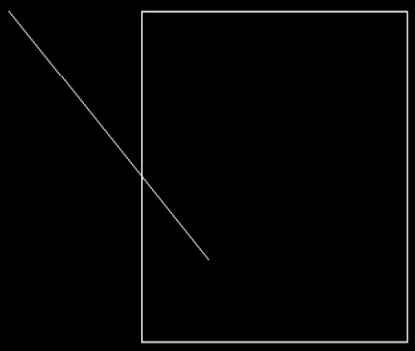
```
        {
                y1=y_max;
        }

        //Vertex 2
        if(x2<x_min)
        {
                x2=x_min;
        }
        if(x2>x_max)
        {
                x2=x_max;
        }
        if(y2<y_min)
        {
                y2=y_min;
        }
        if(y2>y_max)
        {
                y2=y_max;
        }
        setcolor(RED);
        delay(500);
        display();
}
```

```
        else
        {
                cout<<"The line is completelly inside the window\n";
        }
}

void main()
{
        clrscr();
        int gdriver = DETECT,gmode;
        initgraph(&gdriver,&gmode,"C:\\TC\\BGI");
        //initialising the default value for both the array variables
        for(int i=0;i<4;i++)
        {
                flag[i]=0; flag1[i]=0;
        }
        cout<<"Enter the coordiantes for the line\n";
        cout<<"vertex 1: ";
        cin>>x1>>y1;
        cout<<"vertex 2: ";
        cin>>x2>>y2;
        cout<<"Enter the coodinates of the rectangle:\n ";
        cout<<"vertex 1: ";
        cin>>x_min>>y_min;
        cout<<"vertex 2: ";
```

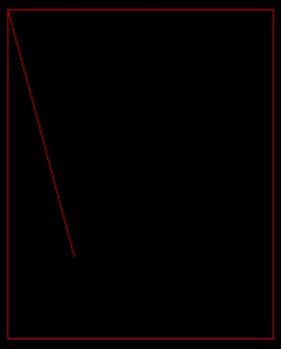
```
        cin>>x_max>>y_max;
        display();
        point_verify1();
        point_verify2();
        verify();
        getch();
}
```

Ouput:

```
Enter the coordiantes for the line
vertex 1: 100 200
vertex 2: 250 350
Enter the coodinates of the rectangle:
 vertex 1: 200 200
vertex 2: 400 400
The line is partially inside the window
```



```
Clipped line to the window is....
```



5. Write a program to clip a polygon using Sutherland Hodgeman algorithm.

Code:

```
#include <graphics.h>
#include <iostream.h>
#include <math.h>
#include <dos.h>
#include <conio.h>
int X[50][50],A[50][50];
int r_x1,r_y1,r_x2,r_y2;
void display(int X[50][50],int n)
{
        setcolor(WHITE);
        rectangle(r_x1,r_y1,r_x2,r_y2);
        int i=0;
        setcolor(YELLOW);
        while(i!=n-1)
        {
                line(X[i][0],X[i][1],X[i+1][0],X[i+1][1]);
                i++;
```

```
        }
        while(i==n-1)
        {
                line(X[i][0],X[i][1],X[0][0],X[0][1]);
                i++;
        }

}
void clipl(int n)
{
        for(int i=0;i<n;i++)
        {
                if(X[i][0]<r_x1)//outside
                {
                A[i][0]=X[i][0];
                A[i][1]=X[i][1];
                X[i][0]=r_x1;
                }

        }
}
```

```
void clipB(int n)
{
	for(int i=0;i<n;i++)
	{
		if(X[i][1]>r_y2)
		{
		A[i][0]=X[i][0];
		A[i][1]=X[i][1];
		X[i][1]=r_y2;
		}
	}
}
void clipR(int n)
{
	for(int i=0;i<n;i++)
	{
		if(X[i][0]>r_x2)
		{
		A[i][0]=X[i][0];
		A[i][1]=X[i][1];
		X[i][1]=r_y2;
```

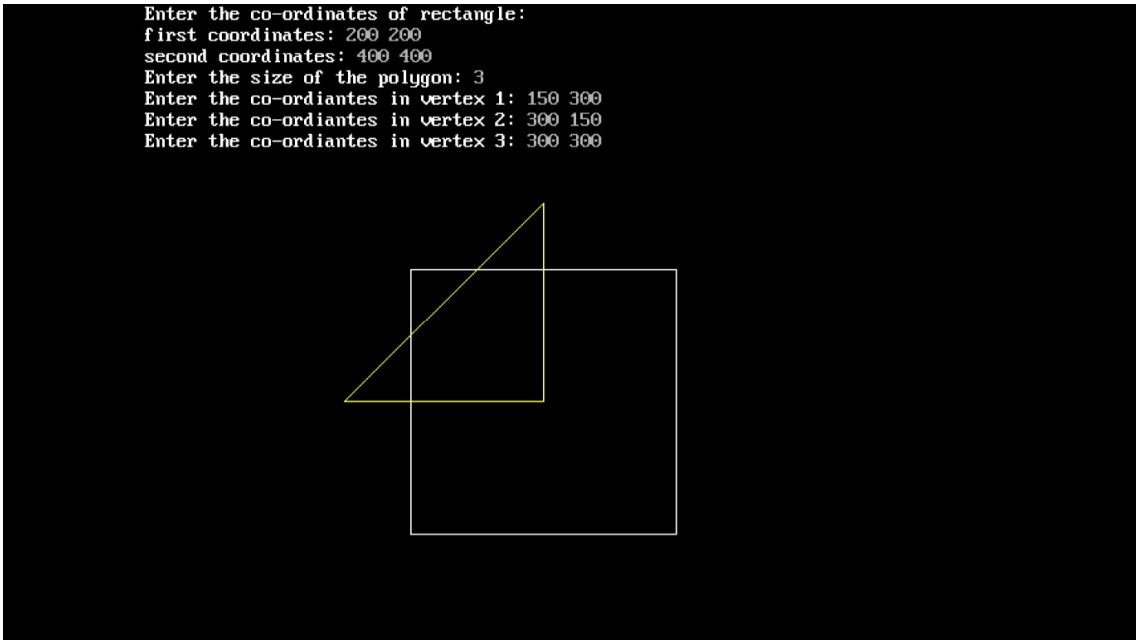
```
                X[i][0]=r_x2;
                }
        }
}

void clipT(int n)
{
        for(int i=0;i<n;i++)
        {
                if(X[i][1]<r_y1)
                {
                A[i][0]=X[i][0];
                A[i][1]=X[i][1];
                X[i][1]=r_y1;
                }
        }
}
void main()
{
clrscr();
int gdriver = DETECT,gmode;
```

```
initgraph(&gdriver,&gmode,"C:\\TC\\BGI");
int n;
cout<<"Enter the co-ordinates of rectangle:\n";
cout<<"first coordinates: ";
cin>>r_x1>>r_y1;
cout<<"second coordinates: ";
cin>>r_x2>>r_y2;
cout<<"Enter the size of the polygon: ";
cin>>n;
        for(int i=0;i<n;i++)
        {
                cout<<"Enter the co-ordiantes in vertex "<<i+1<<": ";
                cin>>X[i][0]>>X[i][1];
        }
        //Original Figure
        display(X,n);
        delay(1000);
        cleardevice();

        //Clipped Left
        cout<<"clipped left\n";
```

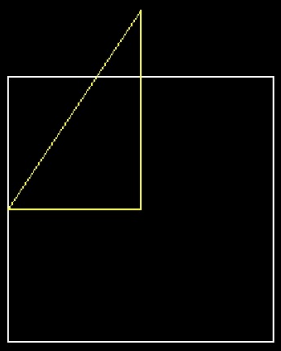
```
  clipl(n);
  display(X,n);
  delay(1000);
  cleardevice();

  //Clipped Bottom
  cout<<"clipped bottom\n";
  clipB(n);
  display(X,n);
  delay(1000);
  cleardevice();

//Clipped right
cout<<"clipped right\n";
  clipR(n);
  display(X,n);
  delay(1000);
  cleardevice();

  //Clipped Top
  cout<<"clipped top\n";
```

```
        clipT(n);
        display(X,n);
        cout<<"Final matrix of polygon is:\n";
        for(i=0;i<n;i++)
        {
        cout<<X[i][0]<<" "<<X[i][1]<<endl;
        }
getch();
}
```

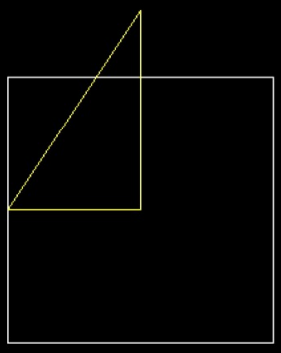
Ouput:



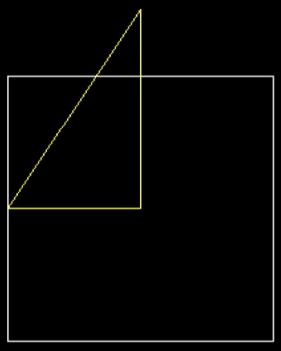
clipped left



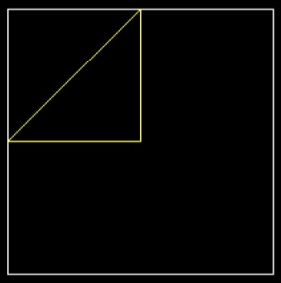
clipped bottom



clipped right



clipped top
Final matrix of polygon is:
200 300
300 200
300 300



6. Write a program to apply various 2D transformations on a 2D object (use homogenous Coordinates).

Code:

```
#include <graphics.h>
#include <iostream.h>
#include <math.h>
#include <dos.h>
#include <conio.h>
void main()
{
clrscr();
float x1,y1,x2,y2,x3,y3;
int gdriver = DETECT,gmode;
initgraph(&gdriver,&gmode,"C:\\TC\\BGI");
cout<<"Enter the co-ordinates of the triangle:\n";
cout<<"Enter the first co-ordinates: ";
cin>>x1>>y1;
cout<<"Enter the second co-ordinates: ";
cin>>x2>>y2;
cout<<"Enter the third co-ordinates: ";
cin>>x3>>y3;
cout<<"Enter your choice: ";
```

```
int i;
line(320,0,320,500);
line(0,250,700,250);
int x11=x1+320, x22=x2+320, x33=x3+320;
int y11=y1+250, y22=y2+250, y33=y3+250;
while(i!=0)
{
cout<<"\nPress: \n1. Translation\n2. Rotation\n3. Scailing\n4. Reflection\n5. Shearing\n0. Exit\n";
cin>>i;
switch(i)
{
	case 1:
	cout<<"Original Figure\n";
	line(x11,y11,x22,y22);
	line(x22,y22,x33,y33);
	line(x33,y33,x11,y11);
	int x,y;
	cout<<"in x direction: ";
	cin>>x;
	cout<<"in y direction: ";
	cin>>y;
	setcolor(YELLOW);
	cout<<"The Resultant figure is in YELLOW COLOR\n";
	line(x11+x,y11+y,x22+x,y22+y);
```

```
line(x22+x,y22+y,x33+x,y33+y);
line(x33+x,y33+y,x11+x,y11+y);
delay(20);
break;

case 2:
cout<<"Original Figure\n";
line(x11,y11,x22,y22);
line(x22,y22,x33,y33);
line(x33,y33,x11,y11);
int angle;
cout<<"Enter the angle for rotation: ";
cin>>angle;
angle= (angle*3.14159265)/180;
int a;
cout<<"Enter:\n 1.Rotation in clockwise direction\n 2. Rotation in anticlockwise direction\n";
cin>>a;
int X1,X2,X3,Y1,Y2,Y3;
switch(a)
{
   case 1:
      X1=(x1*cos(angle))+(y1*sin(angle));
      Y1=(x1*-sin(angle))+(y1*cos(angle));
      X2=(x2*cos(angle))+(y2*sin(angle));
```

```
        Y2=(x2*-sin(angle))+(y2*cos(angle));
        X3=(x3*cos(angle))+(y3*sin(angle));
        Y3=(x3*-sin(angle))+(y3*cos(angle));
        setcolor(GREEN);
        cout<<"The Resultant figure is in GREEN COLOR\n";
        line(X1+320,Y1+250,X2+320,Y2+250);
        line(X2+320,Y2+250,X3+320,Y3+250);
        line(X3+320,Y3+250,X1+320,Y1+250);
        break;
        case 2:
        X1=(x1*cos(angle))+(y1*-sin(angle));
        Y1=(x1*sin(angle))+(y1*cos(angle));
        X2=(x2*cos(angle))+(y2*-sin(angle));
        Y2=(x2*sin(angle))+(y2*cos(angle));
        X3=(x3*cos(angle))+(y3*-sin(angle));
        Y3=(x3*sin(angle))+(y3*cos(angle));
        setcolor(GREEN);
        cout<<"The Resultant figure is in GREEN COLOR\n";
        line(X1+320,Y1+250,X2+320,Y2+250);
        line(X2+320,Y2+250,X3+320,Y3+250);
        line(X3+320,Y3+250,X1+320,Y1+250);
        break;
}
break;
```

```
case 4:
cout<<"original figure:\n";
line(x11,y11,x22,y22);
line(x22,y22,x33,y33);
line(x33,y33,x11,y11);
int ch;

cout<<"Enter:\n1.Reflection in x axis\n2.Reflection in y
axis\n3.Reflection in y=x axis\n4.Reflection in y=-x axis\n";
cin>>ch;
switch(ch)
{
case 1:
int x11=0,x22=0,x33=0,y11=0,y22=0,y33=0;
x11= x1;
y11= -y1;
x22= x2;
y22= -y2;
x33= x3;
y33= -y3;
setcolor(RED);
cout<<"The Resultant figure is in RED COLOR\n";
line(x11+320,y11+250,x22+320,y22+250);
line(x22+320,y22+250,x33+320,y33+250);
line(x33+320,y33+250,x11+320,y11+250);
```

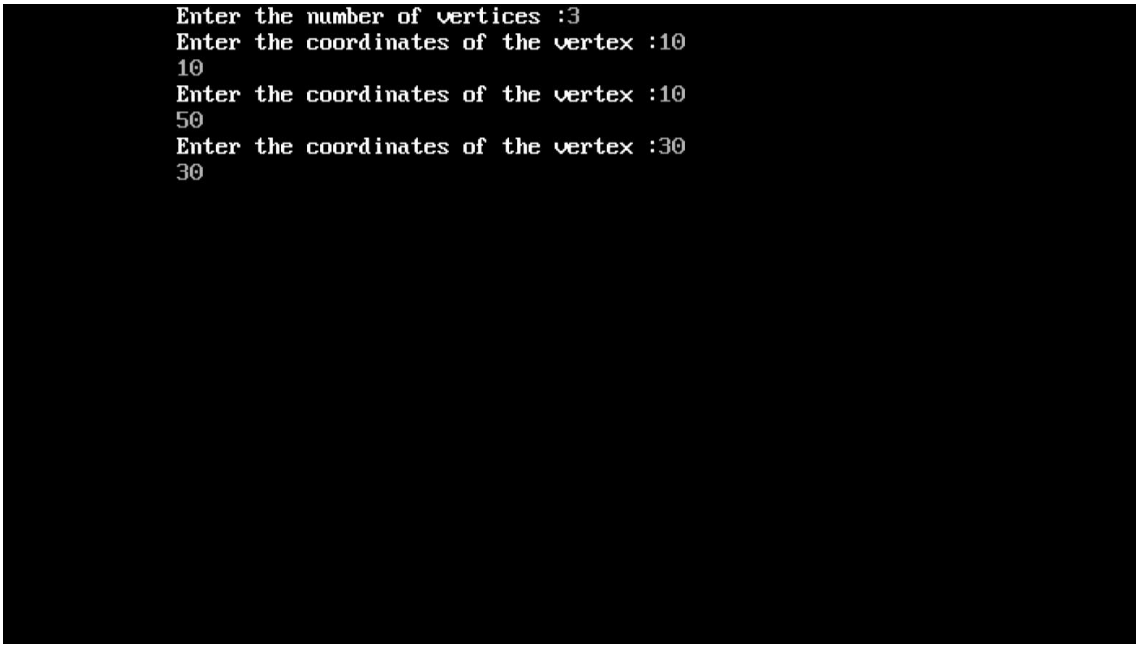
```
break;
case 2:
int xx1=0,xx2=0,xx3=0,yy1=0,yy2=0,yy3=0;
xx1= -x1;
yy1= y1;
xx2= -x2;
yy2= y2;
xx3= -x3;
yy3= y3;
setcolor(RED);
cout<<"The Resultant figure is in RED COLOR\n";
line(xx1+320,yy1+250,xx2+320,yy2+250);
line(xx2+320,yy2+250,xx3+320,yy3+250);
line(xx3+320,yy3+250,xx1+320,yy1+250);
break;
case 3:
int xx11=0,xx22=0,xx33=0,yy11=0,yy22=0,yy33=0;
xx11= x1;
yy11= y1;
xx22= x2;
yy22= y2;
xx33= x3;
yy33= y3;
setcolor(RED);
cout<<"The Resultant figure is in RED COLOR\n";
```

```
line(xx11+320,yy11+250,xx22+320,yy22+250);
line(xx22+320,yy22+250,xx33+320,yy33+250);
line(xx33+320,yy33+250,xx11+320,yy11+250);
break;
case 4:
int xx_1=0,xx_2=0,xx_3=0,yy_1=0,yy_2=0,yy_3=0;
xx_1= -x1;
yy_1= -y1;
xx_2= -x2;
yy_2= -y2;
xx_3= -x3;
yy_3= -y3;
setcolor(RED);
cout<<"The Resultant figure is in RED COLOR\n";
line(xx_1+320,yy_1+250,xx_2+320,yy_2+250);
line(xx_2+320,yy_2+250,xx_3+320,yy_3+250);
line(xx_3+320,yy_3+250,xx_1+320,yy_1+250);
break;
}
break;

case 3:
cout<<"original figure:\n";
line(x11,y11,x22,y22);
line(x22,y22,x33,y33);
```

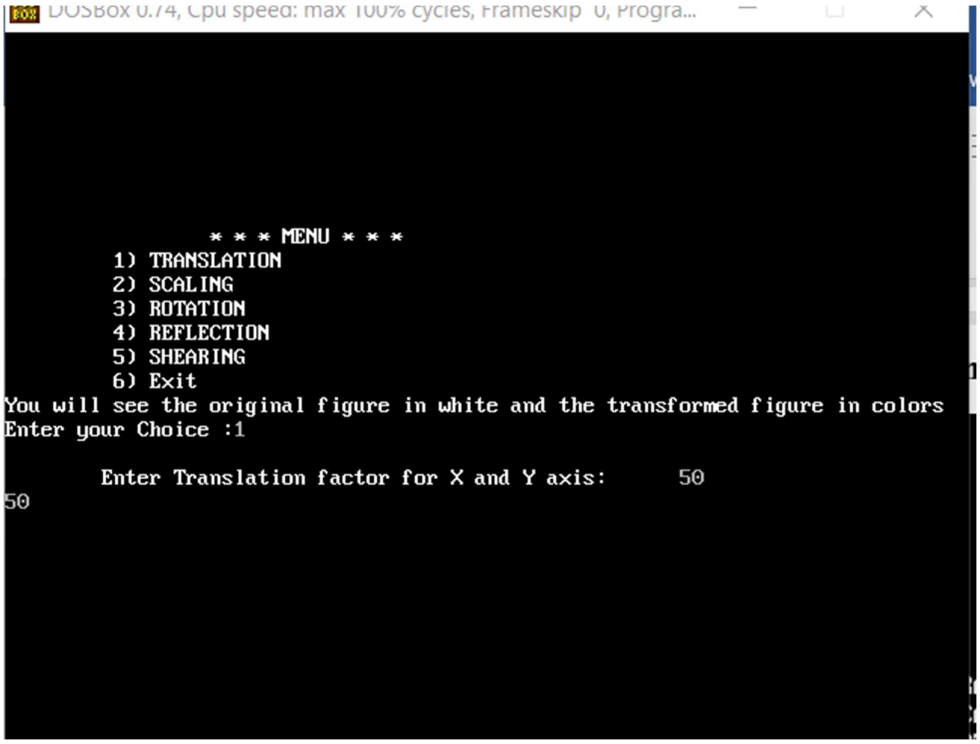
```
line(x33,y33,x11,y11);
int x_s=0,y_s=0;
cout<<"Enter the Scailing Factors in x direction: ";
cin>>x_s;
cout<<"Enter the Scailing Factors in y direction: ";
cin>>y_s;
int x_1,x_2,x_3,y_1,y_2,y_3;
x_1=x1*x_s;
y_1=y1*y_s;
x_2=x2*x_s;
y_2=y2*y_s;
x_3=x3*x_s;
y_3=y3*y_s;
setcolor(BLUE);
cout<<"The Resultant figure is in BLUE COLOR\n";
line(x_1+320,y_1+250,x_2+320,y_2+250);
line(x_2+320,y_2+250,x_3+320,y_3+250);
line(x_3+320,y_3+250,x_1+320,y_1+250);
break;

case 5:
cout<<"Original Figure:\n";
line(x11,y11,x22,y22);
line(x22,y22,x33,y33);
line(x33,y33,x11,y11);
```

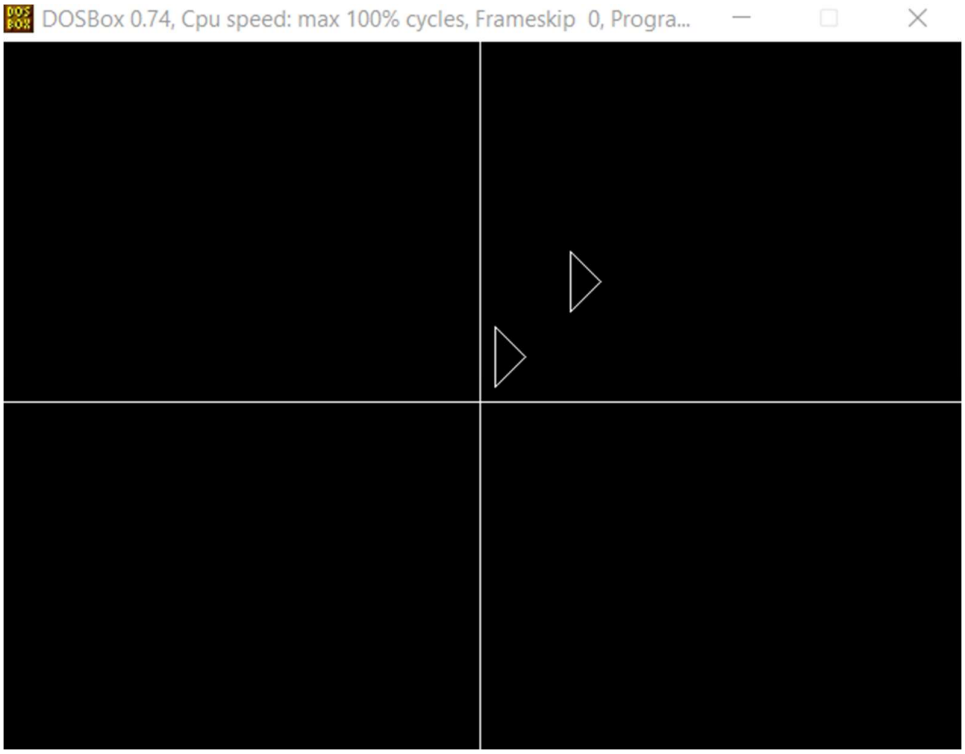
```
        int x_sh, y_sh;
        cout<<"Shearing in x direction: ";
        cin>>x_sh;
        cout<<"Shearing in y direction: ";
        cin>>y_sh;
        int X_1,Y_1,X_2,Y_2,X_3,Y_3;

        X_1= x1+y1*x_sh;
        Y_1= x1*y_sh+y1;
        X_2= x2+y2*x_sh;
        Y_2= x2*y_sh+y2;
        X_3= x3+y3*x_sh;
        Y_3= x3*y_sh+y3;
        setcolor(BROWN);
        cout<<"The Resultant figure is in BROWN COLOR\n";
        line(X_1+320,Y_1+250,X_2+320, Y_2+250);
        line(X_2+320,Y_2+250,X_3+320, Y_3+250);
        line(X_3+320,Y_3+250,X_1+320, Y_1+250);
        break;
}
}
getch();
}
```

OUTPUT:


Enter the number of vertices :3
Enter the coordinates of the vertex :10
10
Enter the coordinates of the vertex :10
50
Enter the coordinates of the vertex :30
30

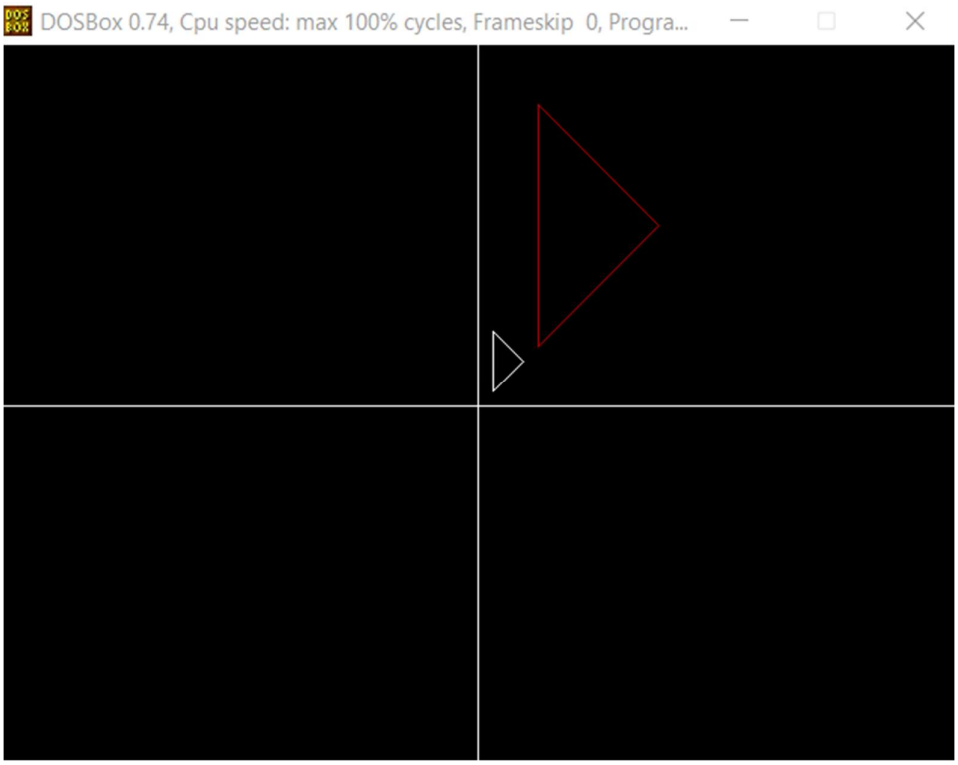
1.Translation



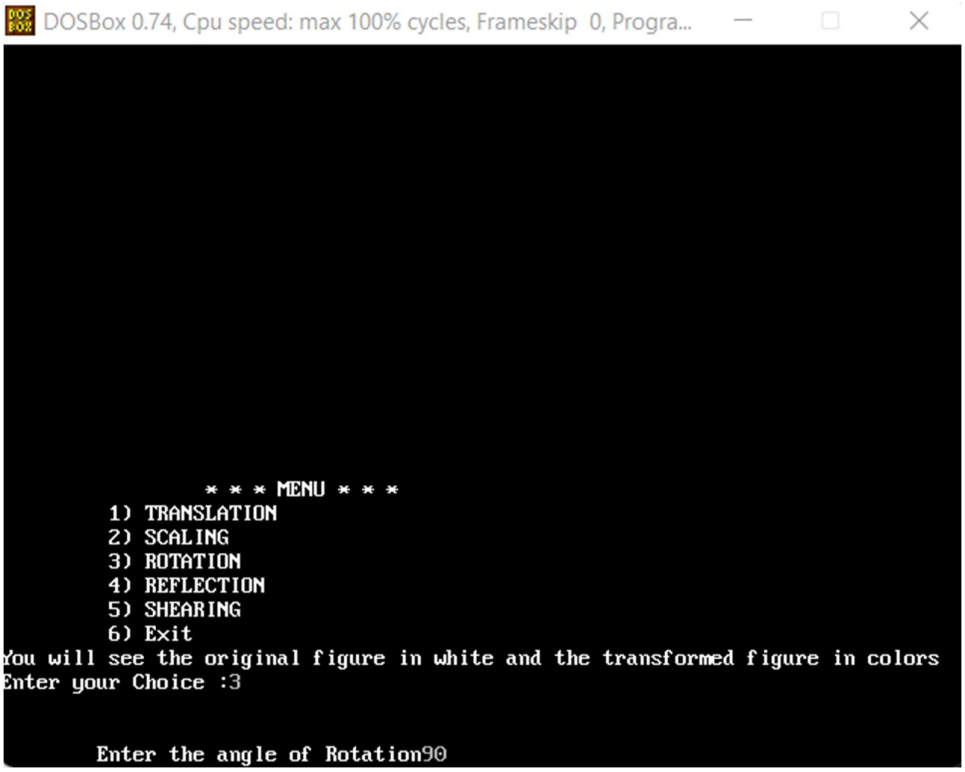


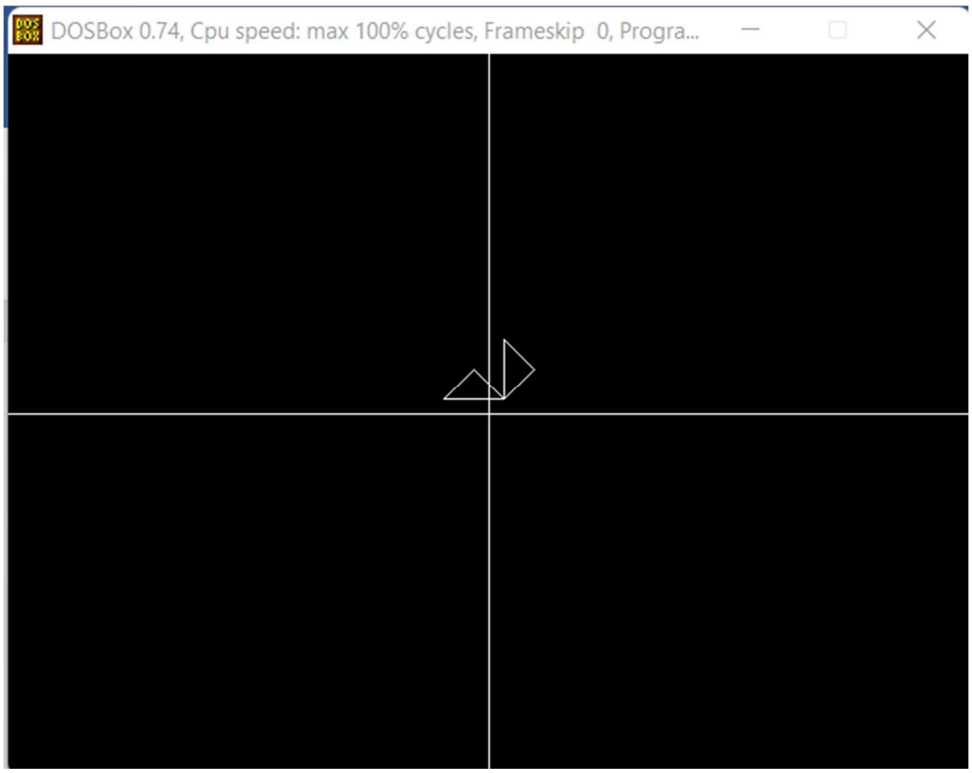
2.Scaling

```
            * * * MENU * * *
        1) TRANSLATION
        2) SCALING
        3) ROTATION
        4) REFLECTION
        5) SHEARING
        6) Exit
You will see the original figure in white and the transformed figure in colors
Enter your Choice :2

       Enter scaling Factor for X and Y axis : 4
4
```



3.Rotation

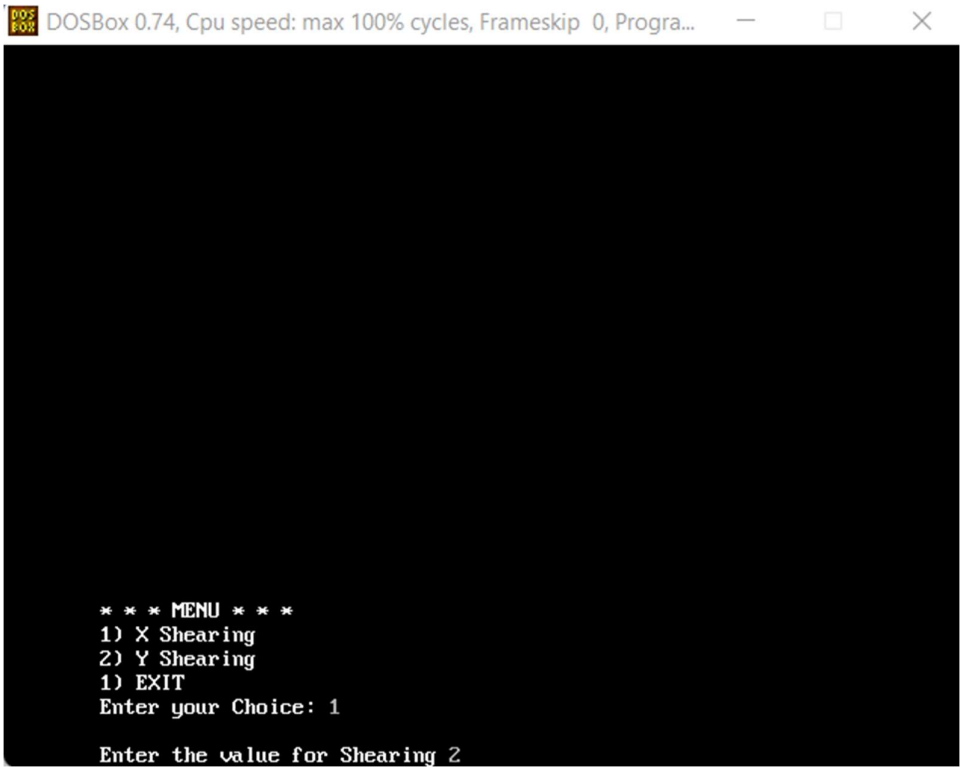

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
* * * MENU * * *
1) TRANSLATION
2) SCALING
3) ROTATION
4) REFLECTION
5) SHEARING
6) Exit
You will see the original figure in white and the transformed figure in colors
Enter your Choice :3
Enter the angle of Rotation90

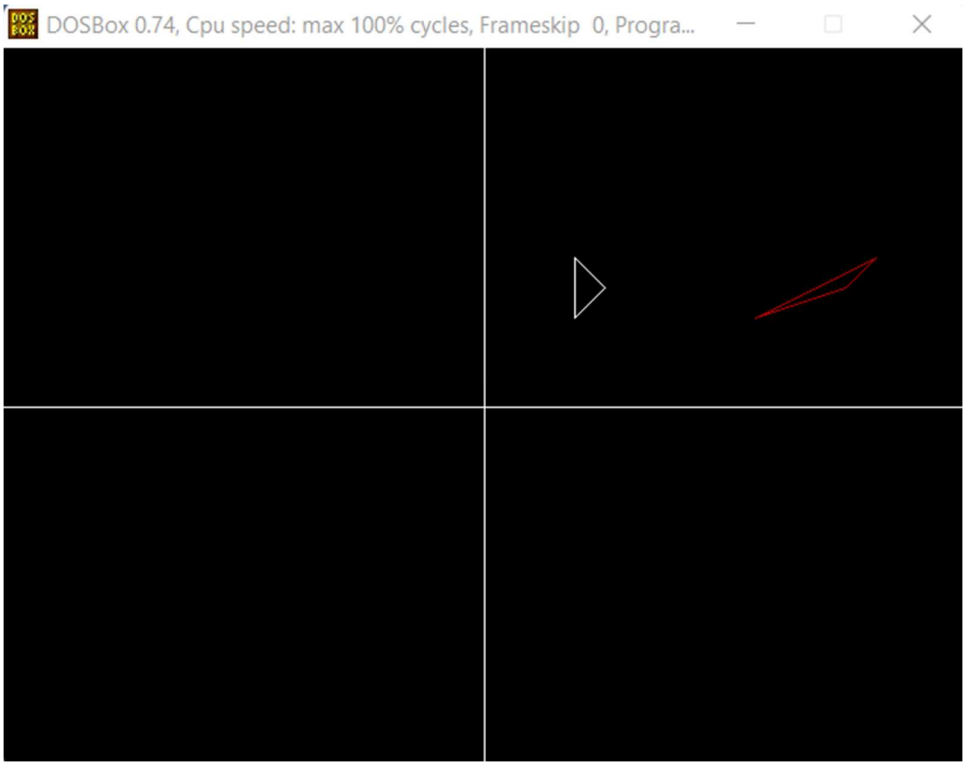

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...

4.Reflection

```
                * * * MENU * * *
          1) TRANSLATION
          2) SCALING
          3) ROTATION
          4) REFLECTION
          5) SHEARING
          6) Exit
You will see the original figure in white and the transformed figure in colors
Enter your Choice :4

1. Ref thru x axis
2. Ref thru y axis
3. Ref thru x=y axis
4. ref thru x=-y axis
5. Ref about origin
 Enter your choice: 1
```

5.Shearing


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
* * * MENU * * *
1) X Shearing
2) Y Shearing
1) EXIT
Enter your Choice: 1
Enter the value for Shearing 2


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...

7. Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.

Code:

```
#include <graphics.h>
#include <iostream.h>
#include <math.h>
#include <dos.h>
#include <conio.h>
int X[50][50],P[50][50];
void display(int n,int b[50][50])
{
	for(int i=0;i<n;i++)
	{
		for(int j=0;j<4;j++)
		{
			cout<<b[i][j]<<" ";
		}
		cout<<endl;
	}
}
void graphics(int n,int mul[50][50])
```

```
{
        int i=0;
        line(320,0,320,500);
        line(0,250,700,250);
        while(i!=n)
        {
                if(mul[i][1]==0)
                {

        line(mul[i][0]+320,mul[i][2]+250,mul[i+1][0]+320,mul[i+1][2]
+250);
                }
                else if(mul[i][0]==0)
                {

        line(mul[i][1]+320,mul[i][2]+250,mul[i+1][1]+320,mul[i+1][2]
+250);
                }
                else
                {

        line(mul[i][1]+320,mul[i][2]+250,mul[i+1][1]+320,mul[i+1][2]
+250);
                }
                i++;
        }
        while(i==n)
```

```
        {
                if(mul[i][1]==0)
                {

        line(mul[i][0]+320,mul[i][2]+250,mul[0][0]+320,mul[0][2]+250
);
                }
                else if(mul[i][0]==0)
                {

        line(mul[i][1]+320,mul[i][2]+250,mul[0][1]+320,mul[0][2]+250
);
                }
                else
                {

        line(mul[i][1]+320,mul[i][2]+250,mul[0][1]+320,mul[0][2]+250
);
                }
                i++;
        }
        getch();
}

void multi(int n, int a[50][50],int b[50][50])
{       //cleardevice();
        int mul[50][50];
```

```
	//Multipliaction of two matrix
	for(int i=0;i<n;i++)
	{
		for(int j=0;j<4;j++)
		{
			mul[i][j]=0;
			for(int k=0;k<4;k++)
			{
				mul[i][j]+=a[i][k]*b[k][j];
			}
		}

	}
	delay(50);
	graphics(n,mul);
}

void projection()
{
	int ch;
	cout<<"\n1. Projection in x direction\n2. Projection in y direction\n3. Projection in z direction\n";
	cin>>ch;
	switch(ch)
	{
```

```
case 1:
for(int i=0;i<4;i++)
{
        for(int j=0;j<4;j++)
        {
                if(i==j&&i!=0)
                {
                P[i][j]=1;
                }
                else
                {
                P[i][j]=0;
                }
        }
}
break;
case 2:
for(i=0;i<4;i++)
{
        for(int j=0;j<4;j++)
        {
                if(i==j&&i!=1)
                {
                P[i][j]=1;
                }
```

```
                else
                {
                P[i][j]=0;
                }
            }
        }
        break;
        case 3:
        for(i=0;i<4;i++)
        {
            for(int j=0;j<4;j++)
            {
                if(i==j&&i!=2)
                {
                P[i][j]=1;
                }
                else
                {
                P[i][j]=0;
                }
            }
        }
        break;
}
```

```
}

void translation(int n,int b[50][50])
{
	//P for the projection matrix;
	int T[50][50],x,y,z;
	cout<<"\nTranslation in X direction\n";
	cin>>x;
	cout<<"Translation in y direection\n";
	cin>>y;
	cout<<"Translation in z direction\n";
	cin>>z;
	for(int i=0;i<n;i++)
	{
		T[i][0]=b[i][0]+x;
		T[i][1]=b[i][1]+y;
		T[i][2]=b[i][2]+z;
		T[i][3]=b[i][3];
	}
	display(n,T);
	projection();
	delay(1500);
	cleardevice();
	multi(n,b,P);
	multi(n,T,P);
```

```
}
void scaling(int n, int b[50][50])
{

        int x_s=0,y_s=0,z_s=0;
        int S[50][50];
        cout<<"Enter the Scaling Factor in x direction: ";
        cin>>x_s;
        cout<<"Enter the Scaling Factor in y direction: ";
        cin>>y_s;
        cout<<"Enter the Scaling Factor in z direction: ";
        cin>>z_s;
        for(int i=0;i<n;i++)
        {
                S[i][0]=b[i][0]*x_s;
                S[i][1]=b[i][1]*y_s;
                S[i][2]=b[i][2]*z_s;
                S[i][3]=b[i][3];
        }
        display(n,S);
        cout<<"Original Figure is in WHITE SHADE\nScaled Figure is
in RED SHADE\n";
        projection();

        delay(1500);
```

```
        cleardevice();
        multi(n,b,P);
        setcolor(RED);
        multi(n,S,P);
}
void rotation(int n, int b[50][50])
{
        int R[50][50],Rx[50][50],Ry[50][50],Rz[50][50];
        int angle;
        cout<<"Enter the angle for rotation: ";
        cin>>angle;
        angle= (angle*3.14159265)/180;
        cout<<angle<<endl;
        int a;
        for(int i=0;i<4;i++)
        {
                for(int j=0;j<4;j++)
                {
                        if(i==j)
                        {
                                Rx[i][j]=1;
                        }
                        else
                        {
                                Rx[i][j]=0;
```

```
            }
        }
}
Rx[1][1]=cos(angle);
Rx[1][2]=sin(angle);
Rx[2][1]=-sin(angle);
Rx[2][2]=cos(angle);
cout<<"Rotation matrix in X direction\n";
display(4,Rx);

// for rotation in y axis
for(i=0;i<4;i++)
{
        for(int j=0;j<4;j++)
        {
                if(i==j)
                {
                        Ry[i][j]=1;
                }
                else
                {
                        Ry[i][j]=0;
                }
        }
}
```

```
Ry[0][0]=cos(angle);
Ry[2][0]=sin(angle);
Ry[0][2]=-sin(angle);
Ry[2][2]=cos(angle);
cout<<"Rotation matrix in Y direction\n";
display(4,Ry);

// for rotation in z axis
for(i=0;i<4;i++)
{
	for(int j=0;j<4;j++)
	{
		if(i==j)
		{
			Rz[i][j]=1;
		}
		else
		{
			Rz[i][j]=0;
		}
	}
}
Rz[0][0]=cos(angle);
Rz[1][0]=-sin(angle);
Rz[1][1]=cos(angle);
```

```
Rz[0][1]=sin(angle);
cout<<"Rotation matrix in Z direction\n";
display(4,Rz);
cout<<"Enter:\n1.Rotation in x direction\n2. Rotation in y direction\n3. Rotation in z direction\n";
cin>>a;
int mul_rot[50][50];
switch(a)
{
case 1:
//Multipliaction of two matrix in x axis
for(int i=0;i<n;i++)
{
    for(int j=0;j<4;j++)
    {
        mul_rot[i][j]=0;
        for(int k=0;k<4;k++)
        {
            mul_rot[i][j]+=b[i][k]*Rx[k][j];
        }
    }

}
projection();
multi(n,b,P);
```

```
  setcolor(RED);
  display(n,mul_rot);
  delay(100);
  graphics(n,mul_rot);
//        multi(n,mul_rot,P);

  break;

  case 2:
  //Multipliaction of two matrix in y axis
  for(i=0;i<n;i++)
  {
        for(int j=0;j<4;j++)
        {
              mul_rot[i][j]=0;
              for(int k=0;k<4;k++)
              {
                    mul_rot[i][j]+=b[i][k]*Ry[k][j];
              }
        }

  }
  projection();
  multi(n,b,P);
  setcolor(RED);
```

```
  display(n,mul_rot);
  delay(100);
  graphics(n,mul_rot);
//        multi(n,mul_rot,P);
  break;

  case 3:
  //Multipliaction of two matrix in z axis
  for(i=0;i<n;i++)
  {
        for(int j=0;j<4;j++)
        {
              mul_rot[i][j]=0;
              for(int k=0;k<4;k++)
              {
                    mul_rot[i][j]+=b[i][k]*Rz[k][j];
              }
        }

  }
  projection();
  multi(n,b,P);
  setcolor(RED);
  display(n,mul_rot);
  delay(100);
```

```
        graphics(n,mul_rot);
        break;
        }



}
void reflection(int n, int b[50][50])
{
        int ref[50][50];
        int ch;
        cout<<"\n1.Reflection in x axis\n2.Reflection in y
axis\n3.Reflection in z axis\n";
        cout<<"\n4.Reflection in x-y axis\n5.Reflection in y-z
axis\n6.Reflection in x-z axis\n";
        cout<<"\n7Reflection in x-y-z axis\n";
        cin>>ch;
        switch(ch)
        {
                case 1://x axis
                for(int i=0;i<n;i++)
                {
                ref[i][0]=-b[i][0];
                ref[i][1]=b[i][1];
                ref[i][2]=b[i][2];
                ref[i][3]=b[i][3];
```

```
}
break;
case 2://y axis
for(i=0;i<n;i++)
{
ref[i][0]=b[i][0];
ref[i][1]=-b[i][1];
ref[i][2]=b[i][2];
ref[i][3]=b[i][3];
}
break;
case 3://z axis
for( i=0;i<n;i++)
{
ref[i][0]=b[i][0];
ref[i][1]=b[i][1];
ref[i][2]=-b[i][2];
ref[i][3]=b[i][3];
}
break;
case 4://x-y axis
for( i=0;i<n;i++)
{
ref[i][0]=-b[i][0];
ref[i][1]=-b[i][1];
```

```
ref[i][2]=b[i][2];
ref[i][3]=b[i][3];
}
break;
case 5: //y-z axis
for( i=0;i<n;i++)
{
ref[i][0]=b[i][0];
ref[i][1]=-b[i][1];
ref[i][2]=-b[i][2];
ref[i][3]=b[i][3];
}
break;
case 6: //x-z axis
for( i=0;i<n;i++)
{
ref[i][0]=-b[i][0];
ref[i][1]=b[i][1];
ref[i][2]=-b[i][2];
ref[i][3]=b[i][3];
}
break;
case 7:// x-y-z axis
for( i=0;i<n;i++)
{
```

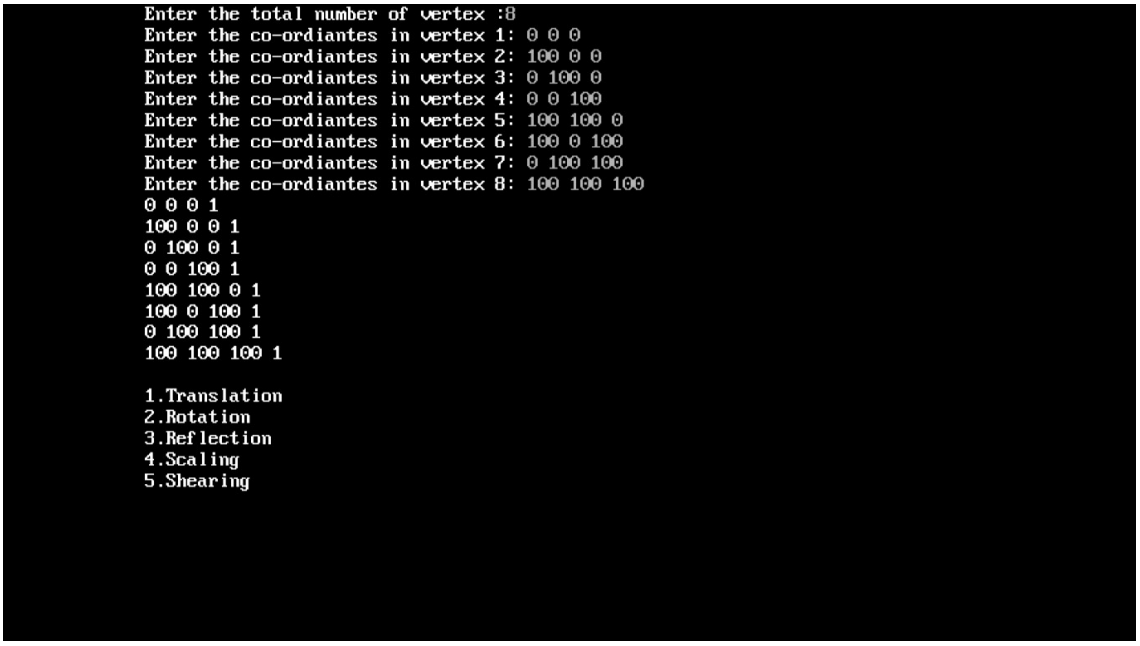
```
                ref[i][0]=-b[i][0];
                ref[i][1]=-b[i][1];
                ref[i][2]=-b[i][2];
                ref[i][3]=b[i][3];
                }
                break;
        }
        projection();
        delay(1500);
        cleardevice();
        multi(n,b,P);
        setcolor(RED);
        delay(100);
        multi(n,ref,P);

}
void shearing(int n,int b[50][50])
{
        int x_sh,y_sh,z_sh,Sh[50][50];
        int ch;
        cout<<"\n1. Shearing in x direction";
        cin>>x_sh;
        cout<<"\n2. Shearing in y direction";
        cin>>y_sh;
        cout<<"\n3. Shearing in z direction\n";
```

```
        cin>>z_sh;
        for(int i=0;i<n;i++)
        {
        Sh[i][0]=b[i][0]+b[i][1]+b[0][2]*x_sh;
        Sh[i][1]=b[i][0]*y_sh+b[i][1]+b[i][2];
        Sh[i][2]=b[i][0]+b[i][2]+b[i][1]*z_sh;
        Sh[i][3]=b[i][3];
        }
        projection();

        delay(1500);
        cleardevice();
        multi(n,b,P);
        setcolor(RED);
        delay(100);
        multi(n,Sh,P);


}

void main()
{
        clrscr();
        int gdriver = DETECT,gmode;
        initgraph(&gdriver,&gmode,"C:\\TC\\BGI");
```

```
int n;;
cout<<"Enter the total number of vertex :";
cin>>n;
for(int i=0;i<n;i++)
{
	cout<<"Enter the co-ordiantes in vertex "<<i+1<<": ";
	cin>>X[i][0]>>X[i][1]>>X[i][2];
	X[i][3]=1;
}
display(n,X);
int choice;
cout<<"\n1.Translation\n2.Rotation\n3.Reflection\n4.Scaling\n5
.Shearing\n";
cin>>choice;
switch(choice)
{
	case 1:
	translation(n,X);
	break;
	case 2:
	rotation(n,X);
	break;
	case 3:
	reflection(n,X);
	break;
```

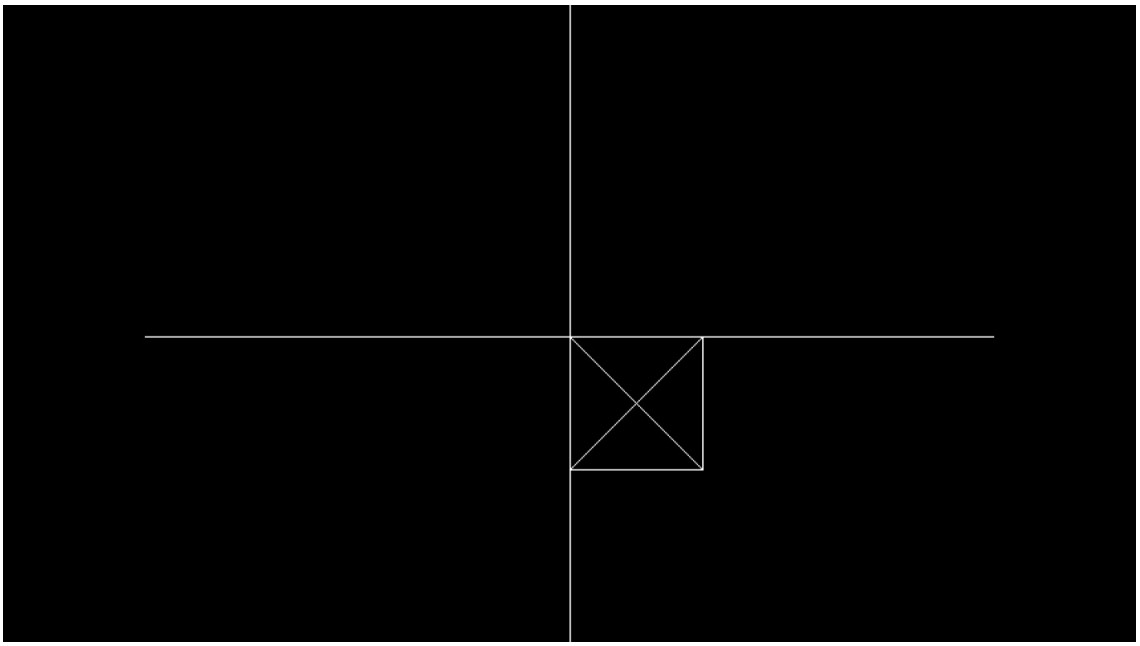
```
            case 4:
            scaling(n,X);
            break;
            case 5:
            shearing(n,X);
            break;
        }
        getch();
}
```

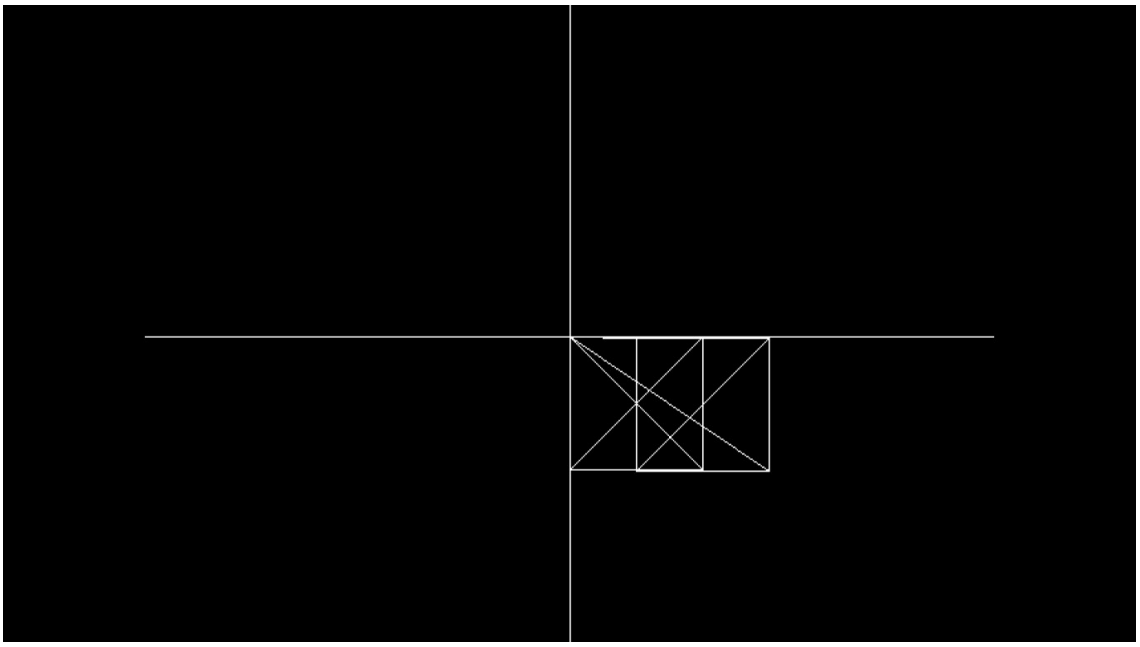
Output:



Translation:

```
100 0 100 1
0 100 100 1
100 100 100 1

1.Translation
2.Rotation
3.Reflection
4.Scaling
5.Shearing
1

Translation in X direction
50
Translation in y direection
2
Translation in z direction
1
50 2 1 1
150 2 1 1
50 102 1 1
50 2 101 1
150 102 1 1
150 2 101 1
50 102 101 1
150 102 101 1

1. Projection in x direction
2. Projection in y direction
3. Projection in z direction
2
```



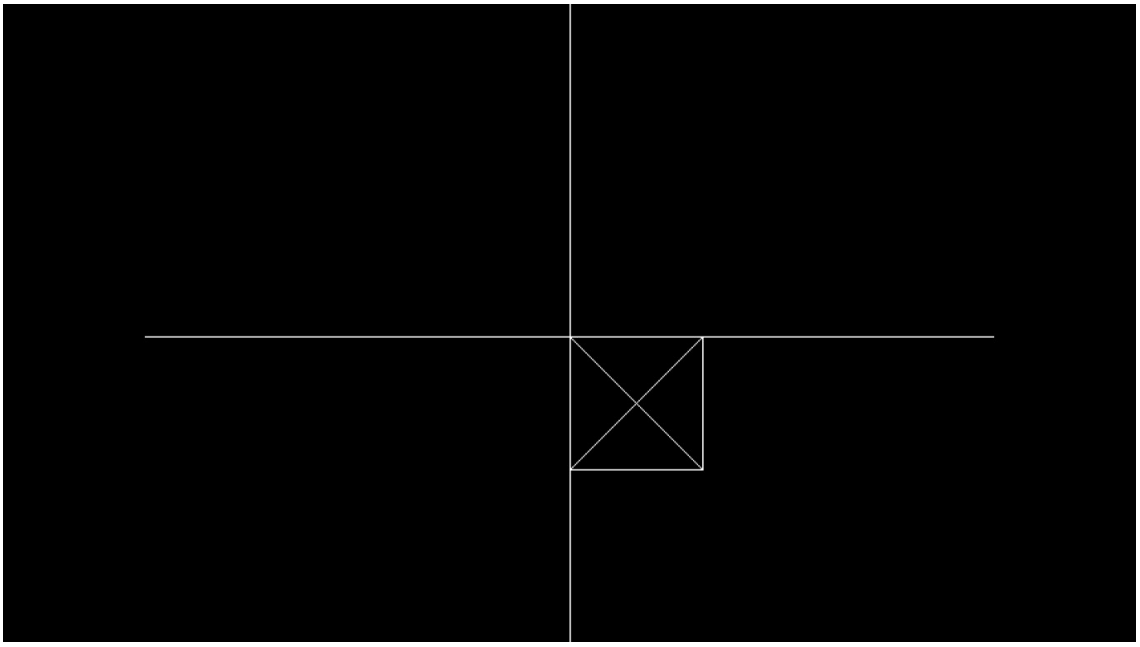


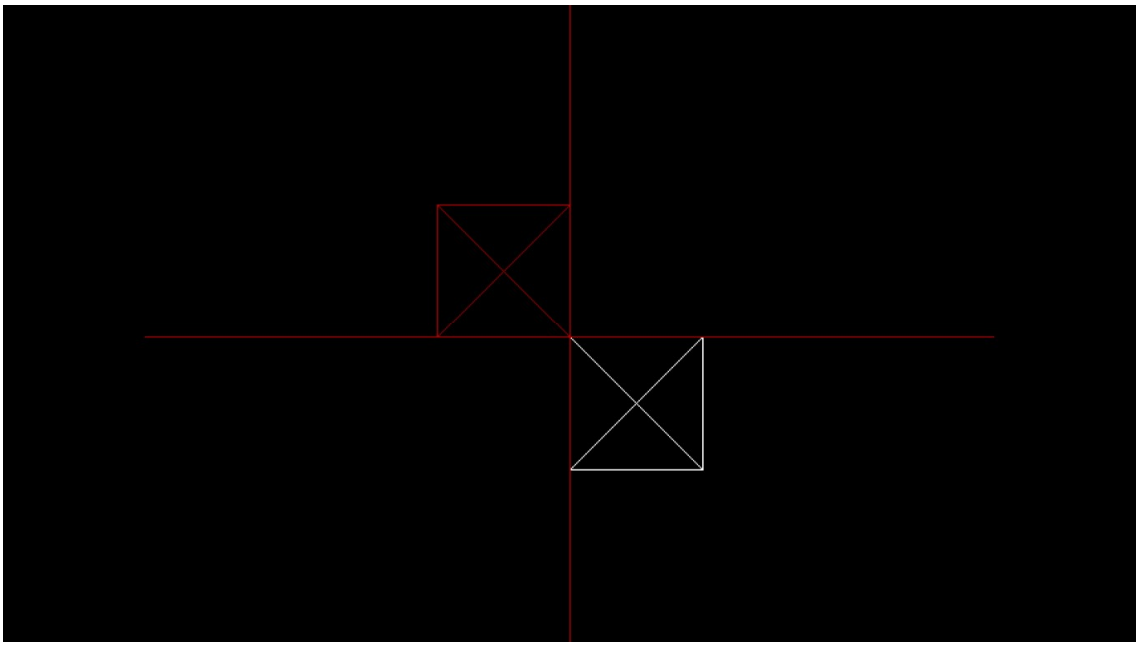
Rotation:

```
5.Shearing
2
Enter the angle for rotation: 50
0
Rotation matrix in X direction
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
Rotation matrix in Y direction
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
Rotation matrix in Z direction
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
Enter:
1.Rotation in x direction
2. Rotation in y direction
3. Rotation in z direction

2

1. Projection in x direction
2. Projection in y direction
3. Projection in z direction
2
```

Reflection:

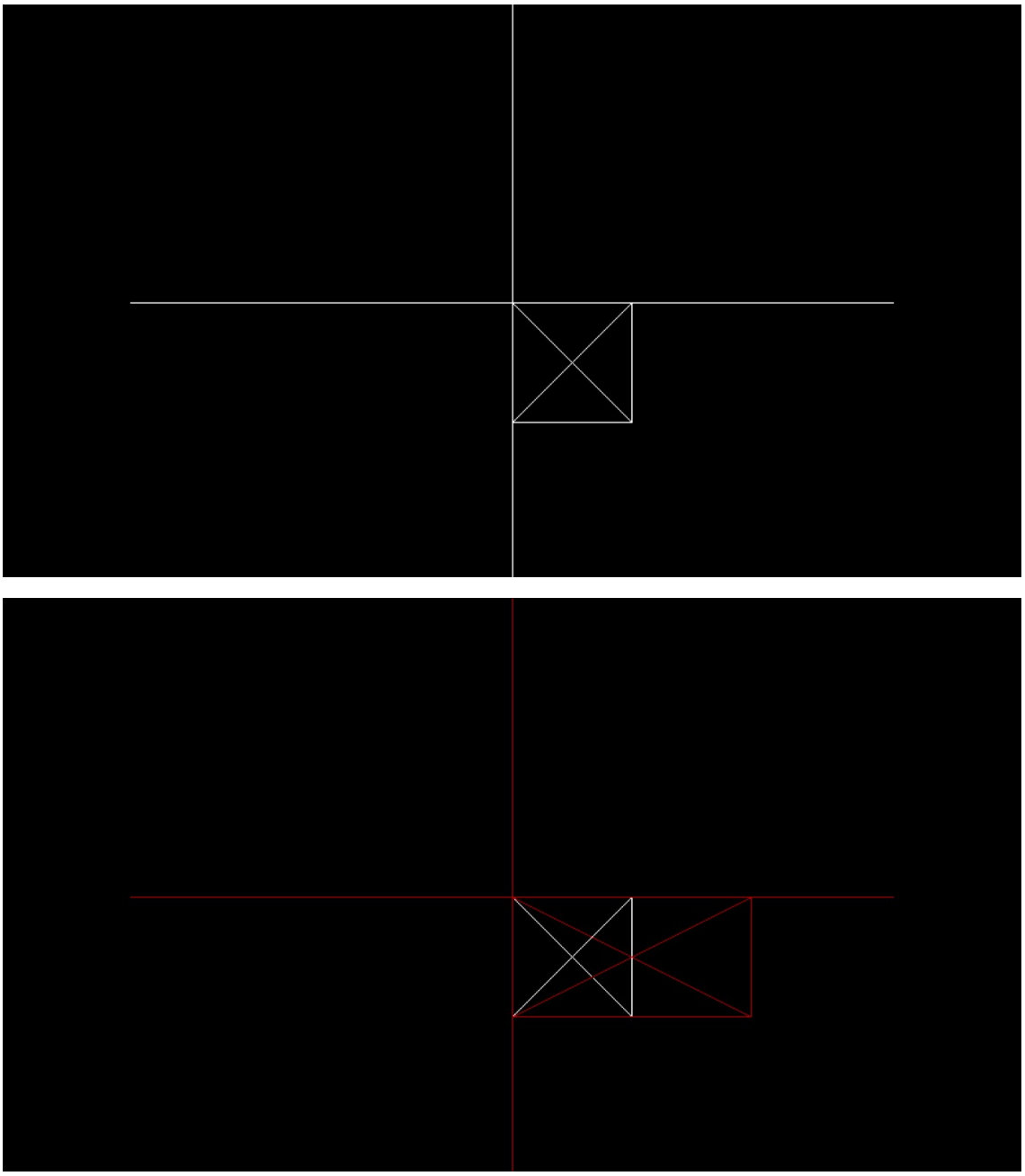
```
100 0 0 1
0 100 0 1
0 0 100 1
100 100 0 1
100 0 100 1
0 100 100 1
100 100 100 1

1.Translation
2.Rotation
3.Reflection
4.Scaling
5.Shearing
3

1.Reflection in x axis
2.Reflection in y axis
3.Reflection in z axis

4.Reflection in x-y axis
5.Reflection in y-z axis
6.Reflection in x-z axis

7Reflection in x-y-z axis
6

1. Projection in x direction
2. Projection in y direction
3. Projection in z direction
2
```





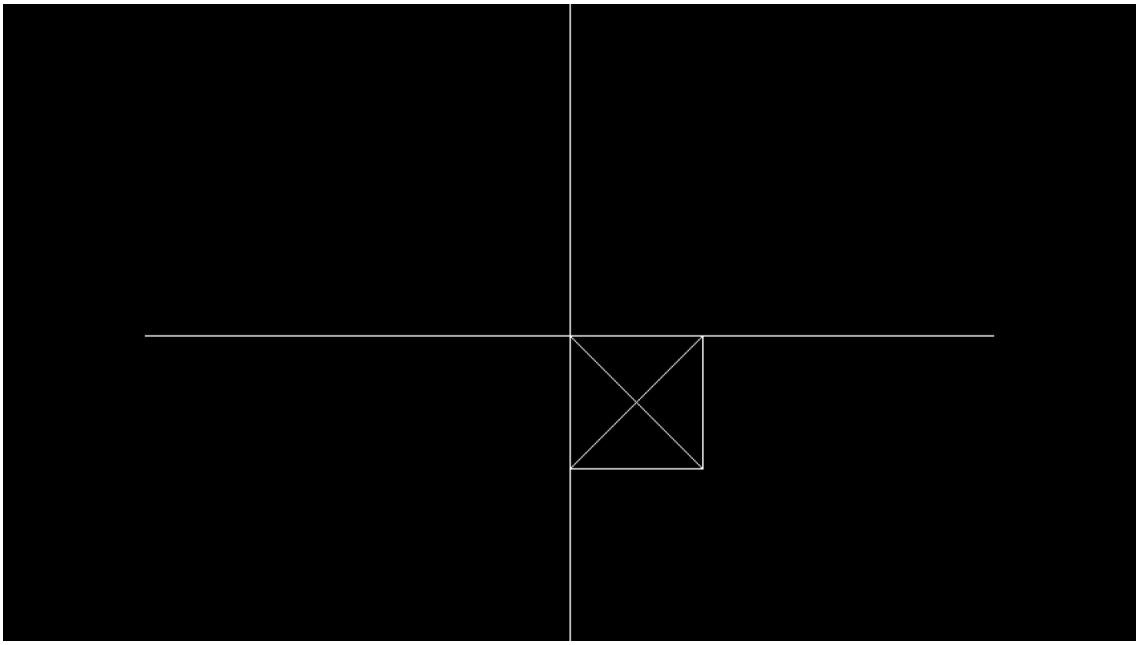
Scaling:

```
0 0 100 1
100 100 0 1
100 0 100 1
0 100 100 1
100 100 100 1

1.Translation
2.Rotation
3.Reflection
4.Scaling
5.Shearing
4
Enter the Scaling Factor in x direction: 2
Enter the Scaling Factor in y direction: 1
Enter the Scaling Factor in z direction: 1
0 0 0 1
200 0 0 1
0 100 0 1
0 0 100 1
200 100 0 1
200 0 100 1
0 100 100 1
200 100 100 1
Original Figure is in WHITE SHADE
Scaled Figure is in RED SHADE

1. Projection in x direction
2. Projection in y direction
3. Projection in z direction
2
```



Shearing:

```
Enter the co-ordiantes in vertex 6: 100 0 100
Enter the co-ordiantes in vertex 7: 0 100 100
Enter the co-ordiantes in vertex 8: 100 100 100
0 0 0 1
100 0 0 1
0 100 0 1
0 0 100 1
100 100 0 1
100 0 100 1
0 100 100 1
100 100 100 1

1.Translation
2.Rotation
3.Reflection
4.Scaling
5.Shearing
5

1. Shearing in x direction20

2. Shearing in y direction1

3. Shearing in z direction
1

1. Projection in x direction
2. Projection in y direction
3. Projection in z direction
2
```



8. Write a program to draw Hermite /Bezier curve.

Code:

```
#include <graphics.h>
#include <iostream.h>
#include <math.h>
#include <dos.h>
#include <conio.h>
void bezier(int x[4],int y[4])
{
        int i;
        double t;
        for(t=0.0;t<1.0;t+=0.0005)
        {
                double xt=pow(1-t,3)*x[0]+3*t*pow(1-
t,2)*x[1]+3*pow(t,2)*(1-t)*x[2]+pow(t,3)*x[3];
                double yt=pow(1-t,3)*y[0]+3*t*pow(1-
t,2)*y[1]+3*pow(t,2)*(1-t)*y[2]+pow(t,3)*y[3];
                putpixel(xt,yt,WHITE);
        }
        for(i=0;i<4;i++)
```

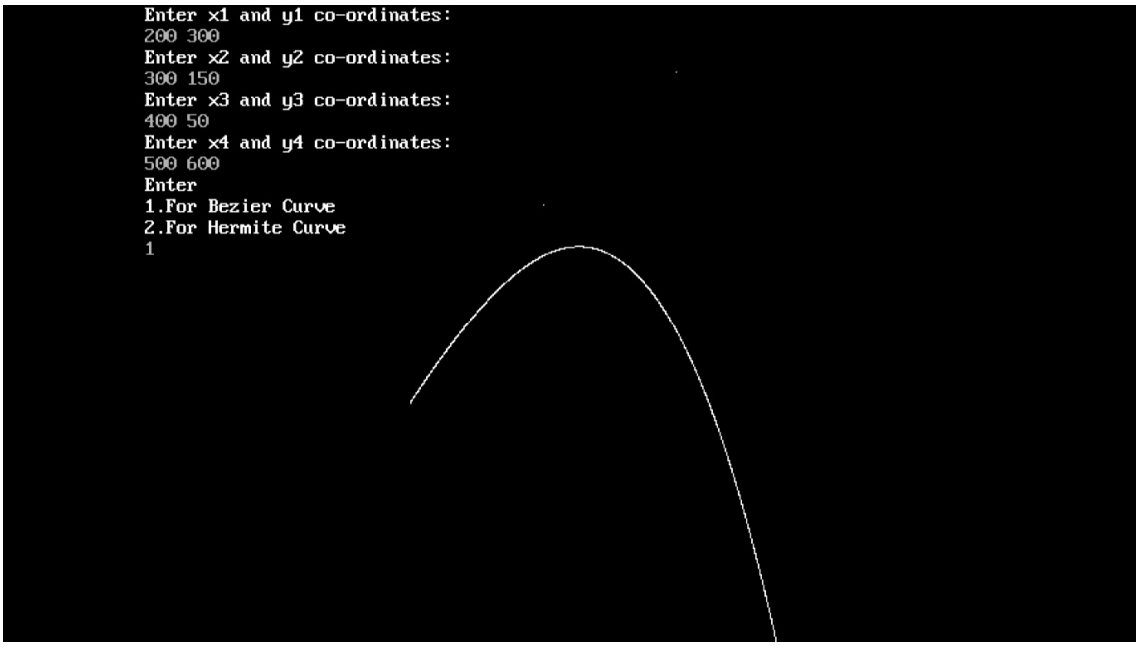
```
        putpixel(x[i],y[i],YELLOW);
        getch();
        closegraph();
        return;
}
void hermite(int x[4], int y[4])
 {
        for(double t = 0; t <= 1; t += 0.001)
        {
              double put_x = (2 * pow(t, 3) - 3 * pow(t, 2) + 1) * x[0] + (-
2 * pow(t, 3) + 3 * pow(t, 2)) * x[1] + (pow(t, 3) - 2 * pow(t, 2) + t) *
x[2] + (pow(t, 3) - pow(t, 2)) * x[3] ;
              double put_y = (2 * pow(t, 3) - 3 * pow(t, 2) + 1) * y[0] + (-
2 * pow(t, 3) + 3 * pow(t, 2)) * y[1] + (pow(t, 3) - 2 * pow(t, 2) + t) *
y[2] + (pow(t, 3) - pow(t, 2)) * y[3];


              putpixel(put_x, put_y, WHITE);
              delay(1);
        }
        for(int i = 0; i <4; i++)
        {
               putpixel(x[i], y[i], 3);
        delay(1);
```

```
    }
        getch();
        closegraph();
        return;

}
void main()
{
        clrscr();
        int gdriver = DETECT,gmode;
        initgraph(&gdriver,&gmode,"C:\\TC\\BGI");
        int x[4],y[4];
        int i;
        for(i=0;i<4;i++)
        {
                cout<<"Enter x"<<i+1<<" and y"<<i+1<<" co-ordinates:
\n";
                cin>>x[i]>>y[i];
        }
        int ch;
        cout<<"Enter\n1.For Bezier Curve\n2.For Hermite Curve\n";
        cin>>ch;
```

```
        switch(ch)
        {
                case 1:
                bezier(x,y);
                break;
                case 2:
                hermite(x,y);
                break;
        }
        getch();
}
```

Output:

Bezier Curve



Hermite Curve