ATMARAM SANATAN DHARMA COLLEGE
UNIVERSITY OF DELHI

Computer Graphics
PRACTICAL FILE

Name- Chakshu Verma

Roll No -19 / 78028

Semester - VI

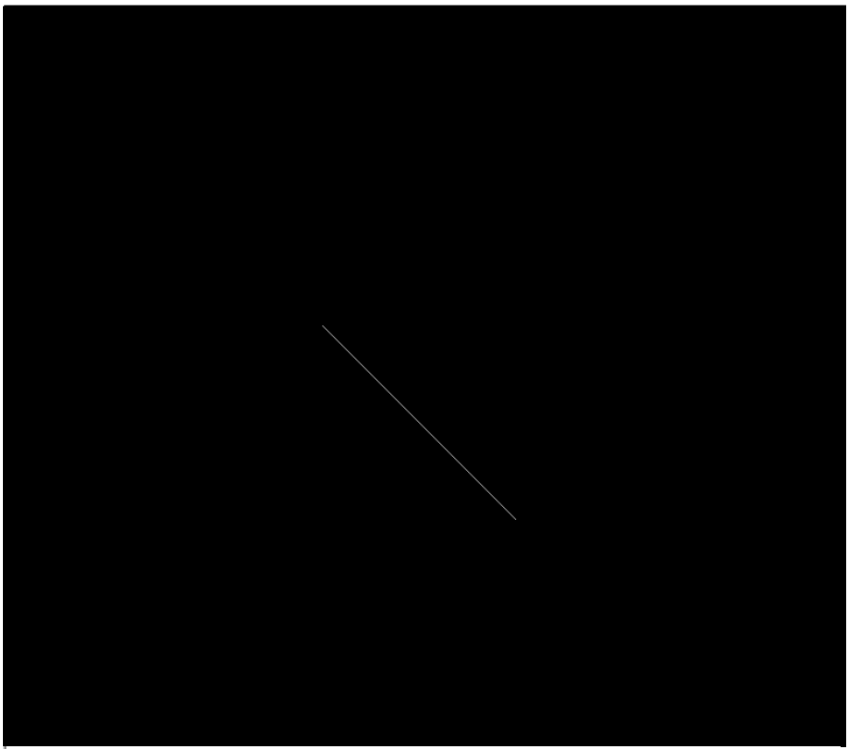
Q1. Write a program to implement DDA and Bresenham's line drawing algorithm.

Code:

```
#include<iostream>
#include<graphics.h>
using namespace std;

void drawline(int x0, int y0, int x1, int y1)
{
   int dx, dy, d, x, y, dE, dNE;
   dx=x1-x0;
   dy=y1-y0;
   x=x0;
   y=y0;
   d=2*dy-dx;
   dE=2*dy;
   dNE=2*(dy-dx);
   while(x<x1)
   {
      if(d>0)
      {
         putpixel(x,y,7);
         y=y+1;
         x=x+1;
         d=d+dNE;
         delay(100);
      }
      else
      {
         putpixel(x,y,7);
         x=x+1;
         d=d+dE;
         delay(100);
      }
   }
}

int main()
{
   int x0, y0, x1, y1;
   int window1 = initwindow(800,800);
```

```
    cout<<"Enter co-ordinates of first point: ";
    cin>>x0>>y0;
    cout<<"Enter co-ordinates of second point: ";
    cin>>x1>>y1;
    drawline(x0, y0, x1, y1);
    closegraph(window1);
    return 0;
}
```

```
Enter co-ordinates of first point: 300
300
Enter co-ordinates of second point: 600
600

--------------------------------
Process exited after 42.02 seconds with return value 0
Press any key to continue . . .
```



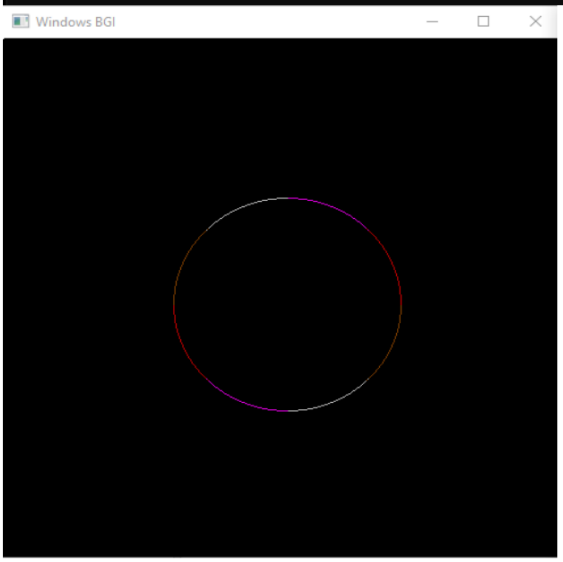
Q2. Write a program to implement mid-point circle drawing algorithm.

Code:

```
#include<iostream>
#include<math.h>
#include<graphics.h>
using namespace std;

void EightSymmetry(int xc, int yc, int x, int y)
{
   putpixel(x+xc,y+yc,7);
   putpixel(y+xc,x+yc,6);
   putpixel(x+xc,-y+yc,5);
   putpixel(-y+xc,x+yc,4);
   putpixel(-x+xc,-y+yc,7);
   putpixel(-y+xc,-x+yc,6);
   putpixel(-x+xc,y+yc,5);
   putpixel(y+xc,-x+yc,4);
}

void drawcircle(int xc, int yc, int r)
{
   int d, x, y;
   x = 0;
   y = r;
   d = 1-r;
   EightSymmetry(xc, yc, x, y);
   while(y>x)
   {
      if(d<0)
      {
         d = d+2*x +3;
         x = x+1;
         EightSymmetry(xc, yc, x, y);
         delay(10);
      }
```

```
        else
        {
            d = d+2*(x-y)+5;
            x = x+1;
            y = y-1;
            EightSymmetry(xc, yc, x, y);
            delay(10);
        }
    }
}

int main()
{
    int xc, yc, r;
    int window1 = initwindow(500,500);
    cout<<"Enter the coordinates of the center of the circle: ";
    cin>>xc>>yc;
    cout<<"Enter the radius of the circle: ";
    cin>>r;
    drawcircle(xc, yc, r);
    delay(5000);
    closegraph(window1);
    return 0;
}
```

```
Enter the coordinates of the center of the circle: 250
250
Enter the radius of the circle: 100
```



Q3. Write a program to clip a line using Cohen and Sutherland line clipping algorithm.

Code

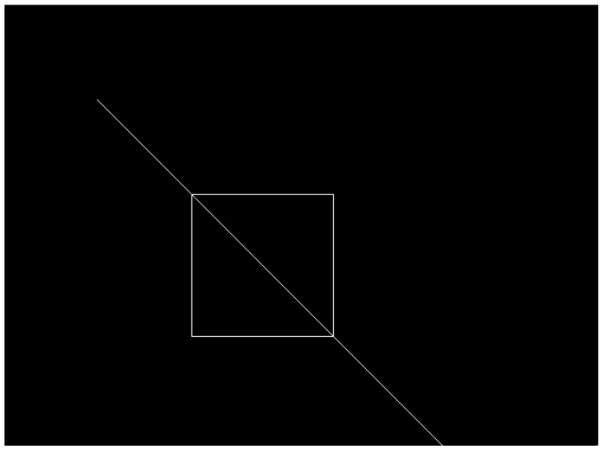
```
#include<iostream>
#include<graphics.h>
using namespace std;
void Window()
{
   line(200,200,350,200);
   line(350,200,350,350);
   line(200,200,200,350);
   line(200,350,350,350);
}
void Code(char c[4],float x,float y)
{
   c[0]=(x<200)?'1':'0';
   c[1]=(x>350)?'1':'0';
   c[2]=(y<200)?'1':'0';
   c[3]=(y>350)?'1':'0';
}
void Clipping (char c[],char d[],float &x,float &y,float m)
{
   int flag=1,i=0;
   for (i=0;i<4;i++)
   {
      if(c[i]!='0' && d[i]!='0')
      {
         flag=0;
         break;
      }
      if(flag)
      {
         if(c[0]!='0')
         {
            y=m*(200-x)+y;
            x=200;
         }
```

```
            else if(c[1]!='0')
            {
               y=m*(350-x)+y;
               x=350;
            }
            else if(c[2]!='0')
            {
               x=((200-y)/m)+x;
               y=200;
            }
            else if(c[3]!='0')
            {
               x=((350-y)/m)+x;
               y=350;
            }
         }
         if(flag==0)
         cout<<"Line lying outside";
      }
}

int main()
{
   int gdriver = DETECT, gmode, errorcode;
   float x1,y1,x2,y2;
   float m;
   char c[4],d[4];
   initgraph(&gdriver, &gmode, NULL);
   cout<<"Enter coordinates";
   cin>>x1>>y1>>x2>>y2;
   cout<<"Before clipping";
   Window();
   delay(1000);
   line(x1,y1,x2,y2);
   delay(2000);
   cleardevice();
   m = float((y2-y1)/(x2-x1));
   Code(c,x1,y1);
   Code(d,x2,y2) ;
   Clipping(c,d,x1,y1,m);
```

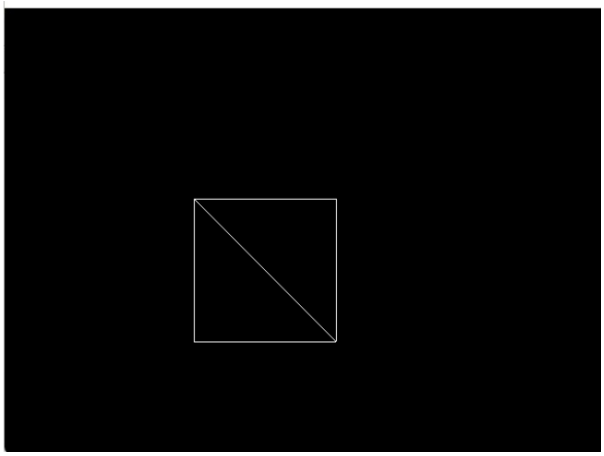
```
    Clipping(d,c,x2,y2,m);
    cout<<"After Clipping";
    Window();
    delay(1000);
    line(x1,y1,x2,y2);
    delay(2000);
    closegraph();
    return 0;
}
```

```
Enter coordinates : 100
100
700
700
```

Before Clipping :-



After Clipping



Q4. Write a program to clip a polygon using Sutherland Hodgeman algorithm.

Code

```
#include<iostream>
#include<graphics.h>
#define round(a) ((int)(a+0.5))
using namespace std;
int xmin=100,xmax=500,ymin=100,ymax=500,arr[20],m;
int k;
void clipleft(int x1,int y1,int x2,int y2)
{
if(x2-x1)
m=(y2-y1)/(x2-x1);
else
m=10000;
if(x1>=xmin && x2>=xmin)
{
arr[k]=x2;
arr[k+1]=y2;
k+=2;
}
if(x1<xmin &&x2>=xmin)
{
arr[k]=xmin;
arr[k+1]=y1+m*(xmin-x1);
arr[k+2]=x2;
arr[k+3]=y2;
k+=4;
}
if(x1>=xmin && x2<xmin)
{
arr[k]=xmin;
arr[k+1]=y1+m*(xmin-x1);
k+=2;
}
}
void cliptop(int x1,int y1,int x2,int y2)
```

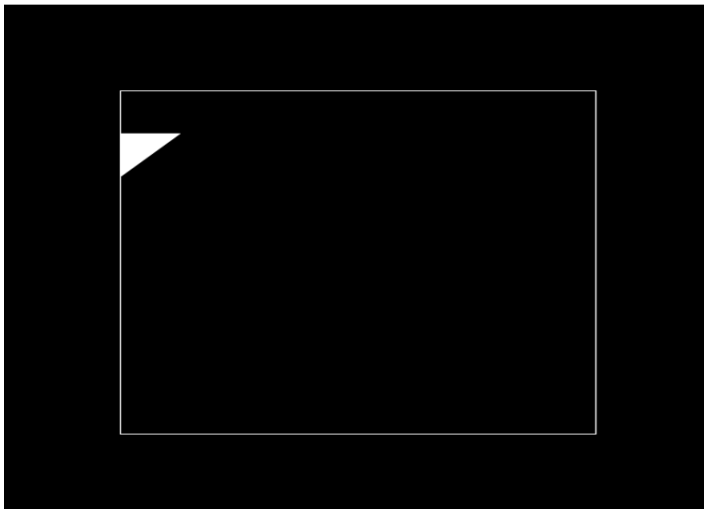
```
{
if(y2-y1)
m=(x2-x1)/(y2-y1);
else
m=10000;
if(y1<=ymax && y2<=ymax)
{
arr[k]=x2;
arr[k+1]=y2;
k+=2;
}
if(y1>ymax && y2<=ymax)
{
arr[k]=x1+m*(ymax-y1);
arr[k+1]=ymax;
arr[k+2]=x2;
arr[k+3]=y2;
k+=4;
}
if(y1<=ymax && y2>ymax)
{
arr[k]=x1+m*(ymax-y1);
arr[k+1]=ymax;
k+=2;
}
}
void clipright(int x1,int y1,int x2,int y2)
{
if(x2-x1)
m=(y2-y1)/(x2-x1);
else
m=10000;
if(x1<=xmax && x2<=xmax)
{
arr[k]=x2;
arr[k+1]=y2;
k+=2;
}
if(x1>xmax &&x2<=xmax)
{
```

```
arr[k]=xmax;
arr[k+1]=y1+m*(xmax-x1);
arr[k+2]=x2;
arr[k+3]=y2;
k+=4;
}
if(x1<=xmax && x2>xmax)
{
arr[k]=xmax;
arr[k+1]=y1+m*(xmax-x1);
k+=2;
}
}
void clipbottom(int x1,int y1,int x2,int y2)
{
if(y2-y1)
m=(x2-x1)/(y2-y1);
else
m=10000;
if(y1>=ymin && y2>=ymin)
{
arr[k]=x2;
arr[k+1]=y2;
k+=2;
}
if(y1<ymin && y2>=ymin)
{
arr[k]=x1+m*(ymin-y1);
arr[k+1]=ymin;
arr[k+2]=x2;
arr[k+3]=y2;
k+=4;
}
if(y1>=ymin && y2<ymin)
{
arr[k]=x1+m*(ymin-y1);
arr[k+1]=ymin ;
k+=2;
}
}
```

```
int main()
{
int polyy[20];
int window1 = initwindow(600,600);
int n,i;
cout<<"Enter the number of edges: ";
cin>>n;
cout<<"Enter the coordinates: ";
for(i=0; i<2*n;i++)
cin>>polyy[i];
polyy[i]=polyy[0];
polyy[i+1]=polyy[1];
rectangle(xmin,ymax,xmax,ymin);
fillpoly(n,polyy);
delay(7000);
cleardevice();
k=0;
for(i=0;i<2*n;i+=2)
clipleft(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
n=k/2;
for(i=0;i<k;i++)
polyy[i]=arr[i];
polyy[i]=polyy[0];
polyy[i+1]=polyy[1];
k=0;
for(i=0;i<2*n;i+=2)
cliptop(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
n=k/2;
for(i=0;i<k;i++)
polyy[i]=arr[i];
polyy[i]=polyy[0];
polyy[i+1]=polyy[1];
k=0;
for(i=0;i<2*n;i+=2)
clipright(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
n=k/2;
for(i=0;i<k;i++)
polyy[i]=arr[i];
polyy[i]=polyy[0];
polyy[i+1]=polyy[1];
```

```
k=0;
for(i=0;i<2*n;i+=2)
clipbottom(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
for(i=0;i<k;i++)
polyy[i]=arr[i];
rectangle(xmin,ymax,xmax,ymin);
if(k)
fillpoly(k/2,polyy);
return 1;
}
```

```
Enter the number of edges: 4
Enter the coordinates:
50 150
150 150
10 300
20 150
```



Q5. Write a program to fill a polygon using Scan line fill algorithm.

Code:

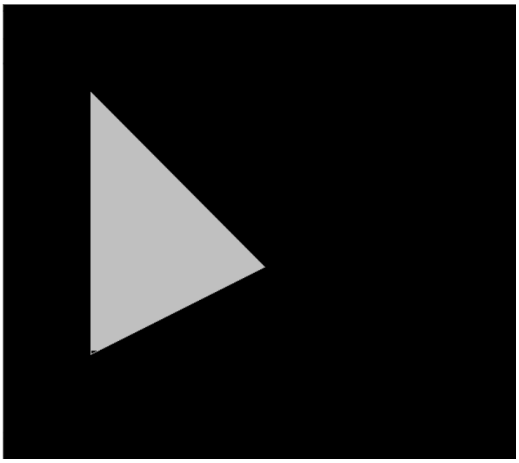
```
#include <conio.h>
#include <iostream>
#include <graphics.h>
#include <stdlib.h>
using namespace std;
//Declaration of class point
class point
{
 public:
 int x,y;
};
class poly
{
 private:
 point p[20];
 int inter[20],x,y;
 int v,xmin,ymin,xmax,ymax;
 public:
 int c;
 void read();
 void calcs();
 void display();
 void ints(float);
 void sort(int);
};
void poly::read()
{
 int i;
 cout<<"POLYGON FILLING SCAN LINE FILL ALGORITHM";
 cout<<"\nEnter the no of vertices of polygon:";
 cin>>v;
 if(v>2)
 {
 for(i=0;i<v; i++) //ACCEPT THE VERTICES
```

```
 {
 cout<<"\nEnter the co-ordinate no.- "<<i+1<<" : ";
 cin>>p[i].x>>p[i].y;
 }
 p[i].x=p[0].x;
 p[i].y=p[0].y;
 xmin=xmax=p[0].x;
 ymin=ymax=p[0].y;
 }
 else
 cout<<"\n Enter valid no. of vertices.";
}
//FUNCTION FOR FINDING
void poly::calcs()
{ //MAX,MIN
 for(int i=0;i<v;i++)
 {
 if(xmin>p[i].x)
 xmin=p[i].x;
 if(xmax<p[i].x)
 xmax=p[i].x;
 if(ymin>p[i].y)
 ymin=p[i].y;
 if(ymax<p[i].y)
 ymax=p[i].y;
 }
}
//DISPLAY FUNCTION
void poly::display()
{
 float s;
s=ymin+0.01;
 delay(100);
 cleardevice();
 while(s<=ymax)
 {
 ints(s);
 sort(s);
 s++;
 }
```

```
}
void poly::ints(float z) //DEFINE FUNCTION INTS
{
 int x1,x2,y1,y2,temp;
 c=0;
 for(int i=0;i<v;i++)
 {
 x1=p[i].x;
 y1=p[i].y;
 x2=p[i+1].x;
 y2=p[i+1].y;
 if(y2<y1)
 {
 temp=x1;
 x1=x2;
 x2=temp;
 temp=y1;
 y1=y2;
 y2=temp;
 }
 if(z<=y2&&z>=y1)
 {
 if((y1-y2)==0)
 x=x1;
 else // used to make changes in x. so that we can fill our polygon after cerain distance
 {
 x=((x2-x1)*(z-y1))/(y2-y1);
 x=x+x1;
 }
 if(x<=xmax && x>=xmin)
 inter[c++]=x;
 }
 }
}
void poly::sort(int z) //SORT FUNCTION
{
 int temp,j,i;
 for(i=0;i<v;i++)
 {
 line(p[i].x,p[i].y,p[i+1].x,p[i+1].y); // used to make hollow outlines of a polygon
```

```
 }
 delay(100);
 for(i=0; i<c;i+=2)
 {
 delay(50);
 line(inter[i],z,inter[i+1],z); // Used to fill the polygon ....
 }
}
int main() //START OF MAIN
{
 initwindow(600,600);
 cleardevice();
 poly x;
 x.read();
 x.calcs();
 cleardevice();
 setcolor(7);
 x.display();
 closegraph(); //CLOSE OF GRAPH
 getch();
 return 0;
}
```

```
POLYGON FILLING SCAN LINE FILL ALGORITHM
Enter the no of vertices of polygon:3

Enter the co-ordinate no.- 1 : 300 300

Enter the co-ordinate no.- 2 : 100 100

Enter the co-ordinate no.- 3 : 100 400
```



Q6. Write a program to apply various 2D transformations on a 2D object (use homogenous Coordinates).

Code

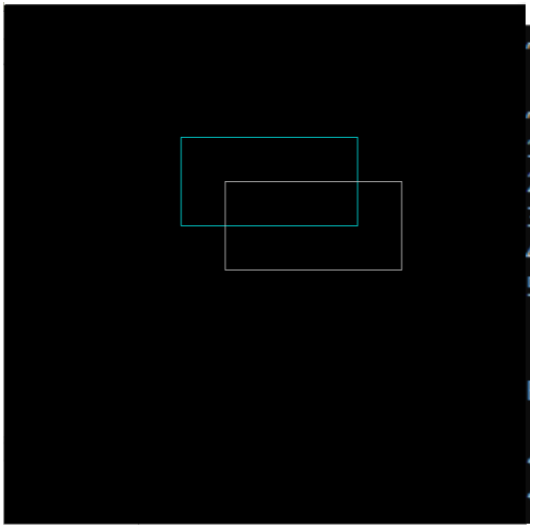
```
#include<iostream>
#include<graphics.h>
#include<math.h>
using namespace std;
int main()
{
int window1 = initwindow(600,600);
cout<<"~~~~~~~~~~~~~~~~~~~~~~~~\n";
cout<<"| MENU |\n";
cout<<"~~~~~~~~~~~~~~~~~~~~~~~~\n";
cout<<"1. Translation\n";
cout<<"2. Rotation\n";
cout<<"3. Scaling\n";
cout<<"4. Reflection\n";
cout<<"5. Shearing\n\n"<<endl;
int ch;
cout<<"Enter your choice: ";
cin>>ch;
switch(ch)
{
case 1:
{
int x1=200, y1=150, x2=400, y2=250;
int tx=50, ty=50;
cout<<"\n<> Rectangle before translation";
setcolor(3);
rectangle(x1, y1, x2, y2);
setcolor(7);
cout<<"\n<> Rectangle after translation";
rectangle(x1+tx, y1+ty, x2+tx, y2+ty);
delay(2000);
break;
```

```
}
case 2:
{
long x1=200,y1=150,x2=400,y2=250;
double a;
cout<<"\n<> Rectangle with rotation";
setcolor(3);
rectangle(x1, y1, x2, y2);
cout<<"\n Enter the angle of rotation: ";
cin>>a;
a = (a*3.14)/180;
long xr = x1 + ((x2-x1)*cos(a) - (y2-y1)*sin(a));
long yr = y1 + ((x2-x1)*sin(a) + (y2-y1)*cos(a));
setcolor(7);
rectangle(x1, y1, xr, yr);
delay(2000);
break;
}
case 3:
{
int x1=100, y1=50, x2=190, y2=90;
int y=2, x=2;
cout<<"\n<> Rectangle before scaling";
setcolor(3);
rectangle(x1, y1, x2, y2);
cout<<"\n<> Rectangle after scaling";
setcolor(7);
rectangle(x1*x, y1*y, x2*x, y2*y);
delay(2000);
break;
}
case 4:
{
int x1=200, y1=300, x2=500, y2=300, x3=350, y3=400;
cout<<"\n<> Triangle before reflection\n";
setcolor(3);
line(x1, y1, x2, y2);
line(x1, y1, x3, y3);
line(x2, y2, x3, y3);
cout<<"\n<> Triangle after reflection\n";
```

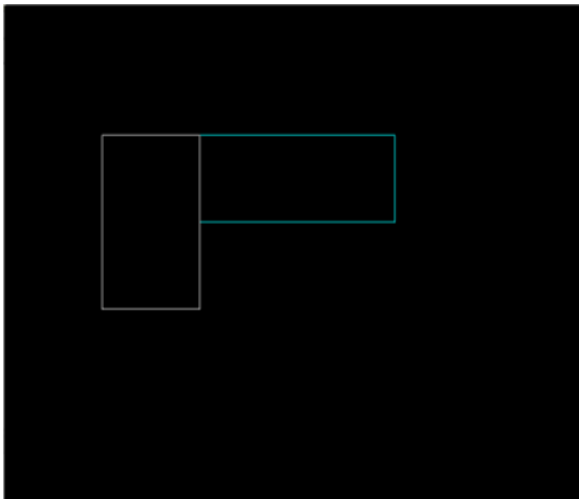
```
setcolor(7);
line(x1, -y1+500, x2, -y2+500);
line(x1, -y1+500, x3, -y3+500);
line(x2, -y2+500, x3, -y3+500);
delay(2000);
break;
}
case 5:
{
int x1=50, y1=100, x2=250, y2=100, x3=50, y3=200, x4=250, y4=200,
shx=1;
cout<<"\n<> Before shearing of rectangle\n";
setcolor(3);
line(x1, y1, x2, y2);
line(x1, y1, x3, y3);
line(x3, y3, x4, y4);
line(x2, y2, x4, y4);
cout<<"\n<> After shearing of rectangle";
x1 = x1 + shx*y1;
x2 = x2 + shx*y2;
x3 = x3 + shx*y3;
x4 = x4 + shx*y4;
setcolor(7);
line(x1, y1, x2, y2);
line(x1, y1, x3, y3);
line(x3, y3, x4, y4);
line(x2, y2, x4, y4);
delay(2000);
break;
}
default:
{
cout<<"Invalid Selection"<<endl;
break;
}
}
delay(5000);
closegraph(window1);
return 0;
}
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~
| MENU |
~~~~~~~~~~~~~~~~~~~~~~~~~~~
1. Translation
2. Rotation
3. Scaling
4. Reflection
5. Shearing
```

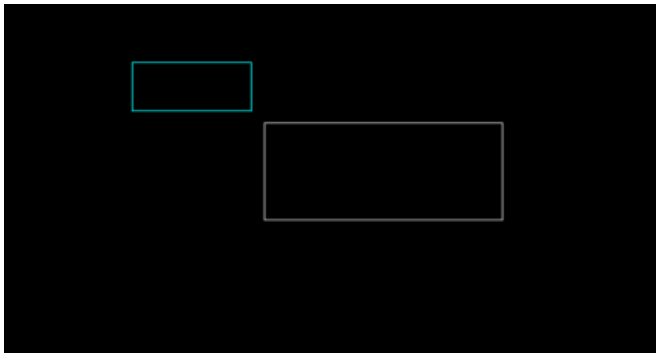
Translation



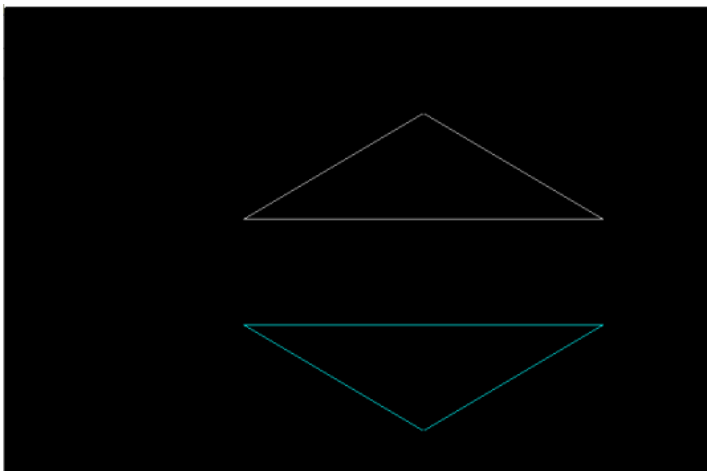
Rotation



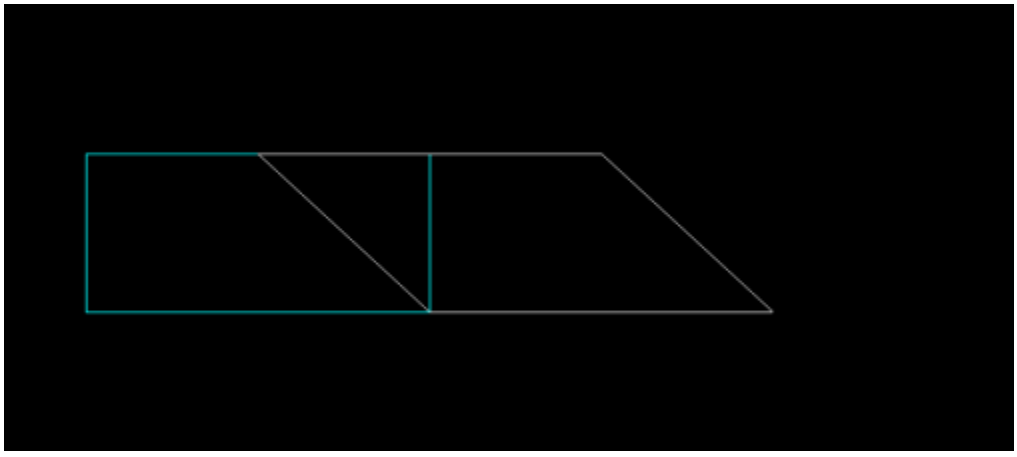
Scaling



Reflection



Shearing

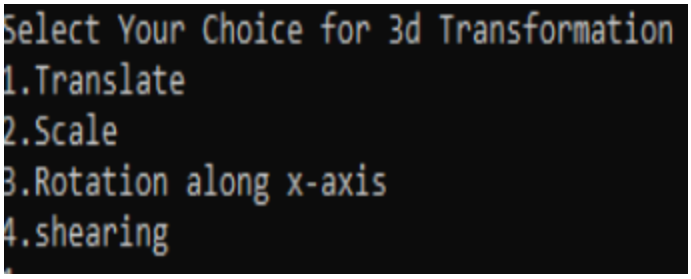


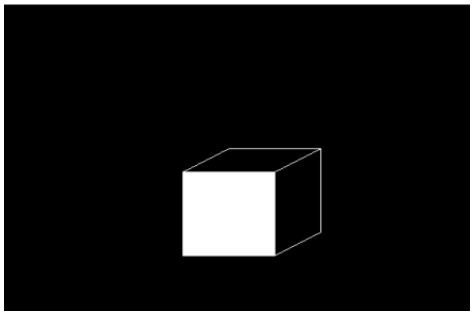
Q7. Write a program to apply various 3D transformations on a 3D object and then apply parallel and perspective projection on it.

Code:

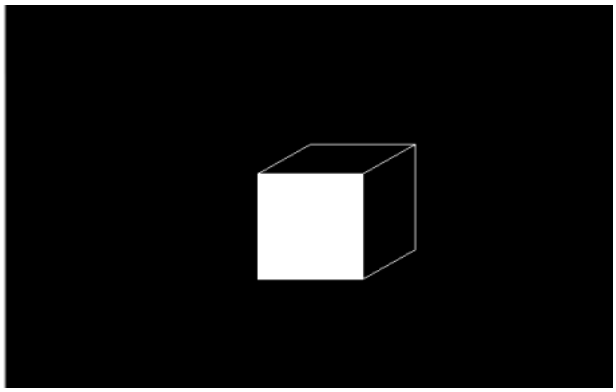
```
#include<iostream>
#include<graphics.h>
#include<cmath>
using namespace std;
int maxx,maxy,midx,midy;
int main()
{
int window1 = initwindow(600,600);
bar3d(200,200,300,300,50,5);
int ch,i,j,k;
int pp[4][4];
cout<<"Select Your Choice for 3d Transformation\n";
cout<<"1.Translate\n2.Scale\n3.Rotation along x-axis\n4.shearing\n";
cin>>ch;
cleardevice();
switch(ch)
{
case 1:
{
int tx,ty;
cout<<"Enter the translation factor for x,y axis"<<endl;
cin>>tx>>ty;
bar3d(200+tx,200+ty,300+tx,300+ty,50,5);
delay(7000);
cleardevice();
outtextxy(10,20,"Parallel projection side view");
bar3d(0,200+ty,0,300+ty,50,5);
delay(7000);
delay(7000);
break;
}
case 2:
```

```
{
int sx,sy;
cout<<"Enter the scaling factor for x,y axis"<<endl;
cin>>sx>>sy;
bar3d(200*sx,200*sy,300*sx,300*sy,50,5);
delay(7000);
cleardevice();
outtextxy(10,20,"Parallel projection side view");
bar3d(0,200*sy,0,300*sy,50,5);
delay(7000);
break;
}
case 3:
{
int ang;
cout<<"Enter the rotation angle"<<endl;
cin>>ang;
ang=(ang*3.14)/180;
int x1= 200*cos(ang)-50*sin(ang);
int y1= 50*cos(ang)+200*sin(ang);
int x2=300*cos(ang)-500*sin(ang);
int y2= 50*cos(ang)+300*sin(ang);
bar3d(x1,y1,x2,y2,50,5);
delay(7000);
break;
}
case 4:
{
int shx,shy;
cout<<"Enter the shearing factor for x,y axis"<<endl;
cin>>shx>>shy;
bar3d(270,200+(shy*270),370,300+(shy*50),50+(270*shx),5);
delay(7000);
break;
}
default: cout<<"Enter valid choice.";
}
return 0;
}
```

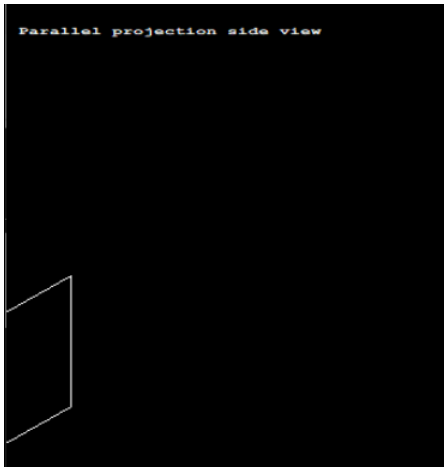




Translation



Parallel Projection(Side View)



Q8. Write a program to draw Hermite /Bezier curve.

Code:

```
#include <iostream>
#include <graphics.h>
#include <cmath>
using namespace std;
int main()
{
int i;
double t,xt,yt;
int window1 = initwindow(500,500);
int ch;
cout<<"Enter the 1 for Bezier Curve and 2 for hermite curve"<<endl;
cin>>ch;
switch(ch)
{
case 1:
{
int x[4]={400,300,400,450};
int y[4]={400,350,275,300};
outtextxy(50,50,"Bezier Curve");
for(t=0;t<=1;t=t+0.0005){
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];
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);
delay(4000);
}
break;
}
case 2:
{
```

```
int x1[4]={200,100,200,250};
int y1[4]={200,150,75,100};
outtextxy(50,50,"Hermite Curve");
for(t=0;t<=1;t=t+0.00001){
xt=x1[0]*(2*pow(t,3)-(3*t*t)+1)+x1[1]*(-
2*pow(t,3)+(3*t*t))+x1[2]*(pow(t,3)-(2*t*t)+t)+x1[3]*(pow(t,3)-(t*t));
yt=y1[0]*(2*pow(t,3)-(3*t*t)+1)+y1[1]*(-
2*pow(t,3)+(3*t*t))+y1[2]*(pow(t,3)-(2*t*t)+t)+y1[3]*(pow(t,3)-(t*t));
putpixel (xt, yt,WHITE);
}
for (i=0; i<4; i++)
{
putpixel (x1[i], y1[i], YELLOW);
delay(9000);
}
break;
}
default: cout<<"Enter valid choice.";
}
return 4;
}
```

Bezier Curve

Hermit Curve