Name of the program: B.Sc. (H) Computer Science

Semester: VI

Paper Name: Computer Graphics

Paper Code: 32341602

Name: SNEHA RANI

College Roll Number: 2020350

Examination Roll Number: 20066570052

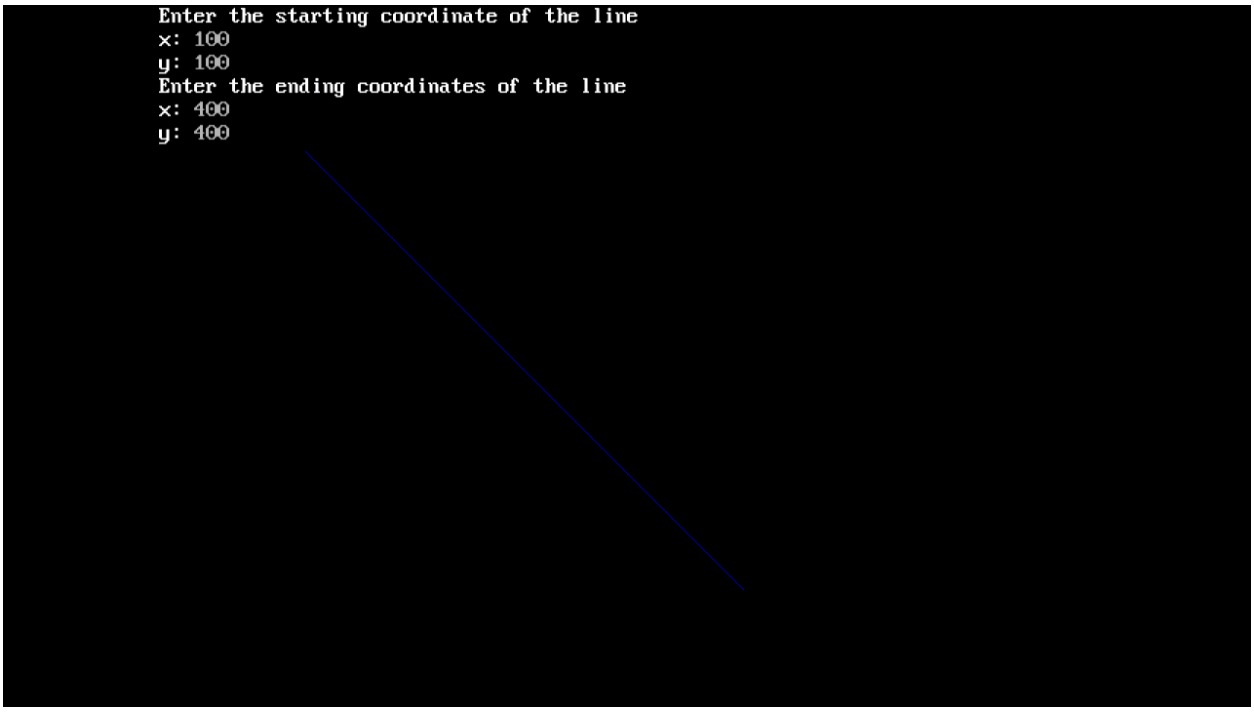
Q1. Write a program to implement DDA and Bresenham's line drawing algorithm.

DDA ALGORITHM

```
#include<iostream.h>
#include<graphics.h>
#include<conio.h>
 void dda(int x1,int y1,int x2, int y2 )
{
 int dx=0,dy=0,m=0,x=0,y=0;
 dy=y2-y1;
 dx=x2-x1;
 m=dy/dx;
 y=y1;
 for(x=x1;x<x2;x++)
 {
  putpixel(x,y,BLUE);
  y=y+m;
 }
}
void main()
{
 int gd= DETECT,gm;
 initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
 int x1,x2,y1,y2;
 x1=x2=y1=y2=0;
```

```
 cout<<"Enter the starting coordinate of the line\n";
 cout<<"x: ";
 cin>>x1;
 cout<<"y: ";
 cin>>y1;
 cout<<"Enter the ending coordinates of the line\n";
 cout<<"x: ";
 cin>>x2;
 cout<<"y: ";
 cin>>y2;
 dda(x1,y1,x2,y2);
 getch();
}
```

OUTPUT

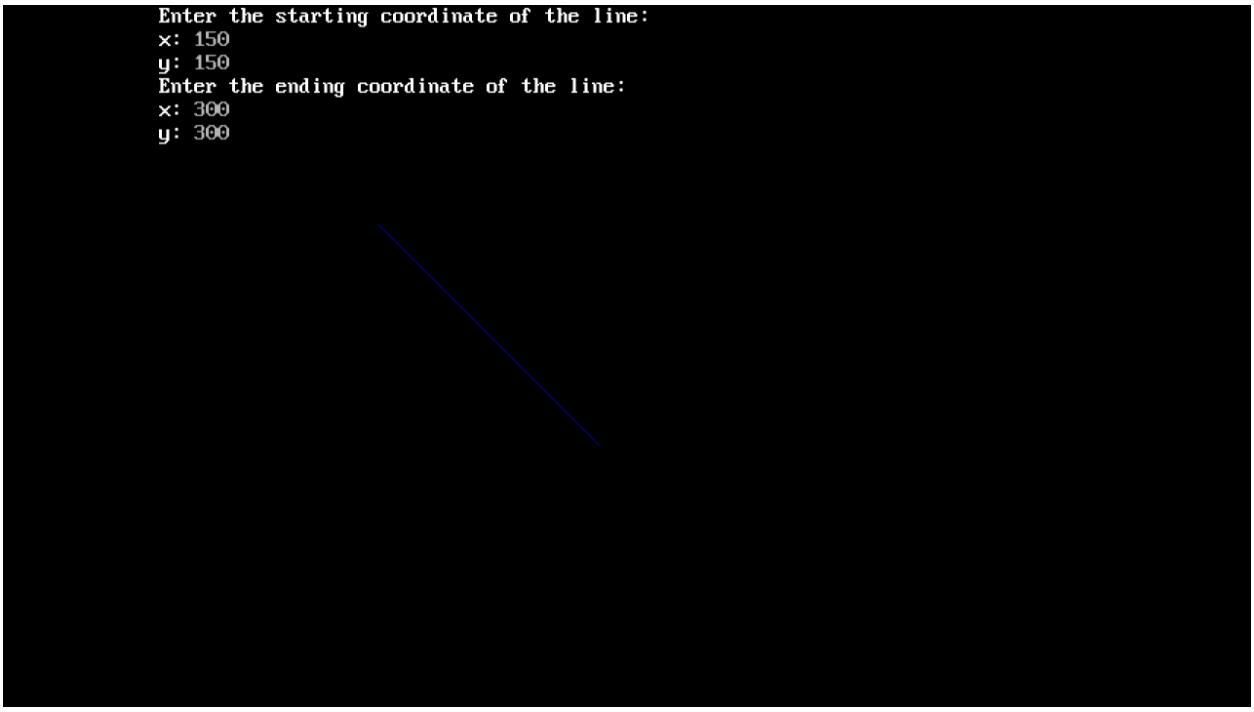

Enter the starting coordinate of the line
x: 100
y: 100
Enter the ending coordinates of the line
x: 400
y: 400

Bresenham's ALGORITHM

```
#include<iostream.h>
#include<graphics.h>
#include<conio.h>
void MidpointLine(int x1,int y1,int x2,int y2)
{
int dx=x2-x1;
int dy=y2-y1;
int d=2*dy-dx;
int incrE=2*dy;
int incrNE=2*(dy-dx);
int x=x1;
int y=y1;
putpixel(x,y,BLUE);
while(x<x2)
{
 if(d<=0)
 {
  d=d+incrE;
  x++;
 }
 else
 {
  d=d+incrNE;
  x++;
```

```
  y++;
 }
 putpixel(x,y,BLUE);
}
}
void main()
{
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
int x1,x2,y1,y2;
x1=x2=y1=y2=0;
cout<<"Enter the starting coordinate of the line:\n";
cout<<"x: ";
cin>>x1;
cout<<"y: ";
cin>>y1;
cout<<"Enter the ending coordinate of the line:\n";
cout<<"x: ";
cin>>x2;
cout<<"y: ";
cin>>y2;
MidpointLine(x1,y1,x2,y2);
getch();
}
```

OUTPUT


Enter the starting coordinate of the line:
x: 150
y: 150
Enter the ending coordinate of the line:
x: 300
y: 300

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

```
#include<iostream.h>
#include<graphics.h>
#include<conio.h>
/*Center of circle is inputed from user
 not assuming that center is (0,0)*/

void circlepoints(int x,int y, int cx,int cy)
{
 putpixel(x+cx,y+cy,YELLOW);
 putpixel(y+cx,x+cy,YELLOW);
 putpixel(y+cx,-x+cy,YELLOW);
 putpixel(x+cx,-y+cy,YELLOW);
 putpixel(-x+cx,-y+cy,YELLOW);
 putpixel(-y+cx,-x+cy,YELLOW);
 putpixel(-y+cx,x+cy,YELLOW);
 putpixel(-x+cx,y+cy,YELLOW);
}
void midpointcircle(int cx,int cy,int r)
{
  int x,y,D;
  x=0;
  y=r;
  circlepoints(x,y,cx,cy);
```

```
 D=(5/4)-r;


 while(y>x)
 {
  if(D<=0)
  {
    D+=(2*x)+3;
  }
  else
  {
    D+=(2*x)-(2*y)+5;
    y--;
  }
  x++;
  circlepoints(x,y,cx,cy);
 }
}
void main()
{
 int gd= DETECT,gm;
 initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
 int cx,cy,r;
 cx=cy=r=0;
 cout<<"Enter the Center (x) coordinate : ";
 cin>>cx;
```

```
 cout<<"Enter the Center (y) coordinate : ";
 cin>>cy;
 cout<<"Enter Radius of the circle : ";
 cin>>r;
 midpointcircle(cx,cy,r);
 getch();
 closegraph();
}
```

OUTPUT

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

```
#include<iostream.h>
#include<graphics.h>
#include<conio.h>
int compOutCode(
double x,double y,
double xmin,double xmax,
double ymin,double ymax)
{
int code=0000;
if(y>ymax)
code+=0001;
else if(y<ymin)
code+=0010;
if(x>xmax)
code+=0100;
else if(x<xmin)
code+=1000;

 return code;
}
void cohen(
double x0,double y0,
double x1, double y1,
```

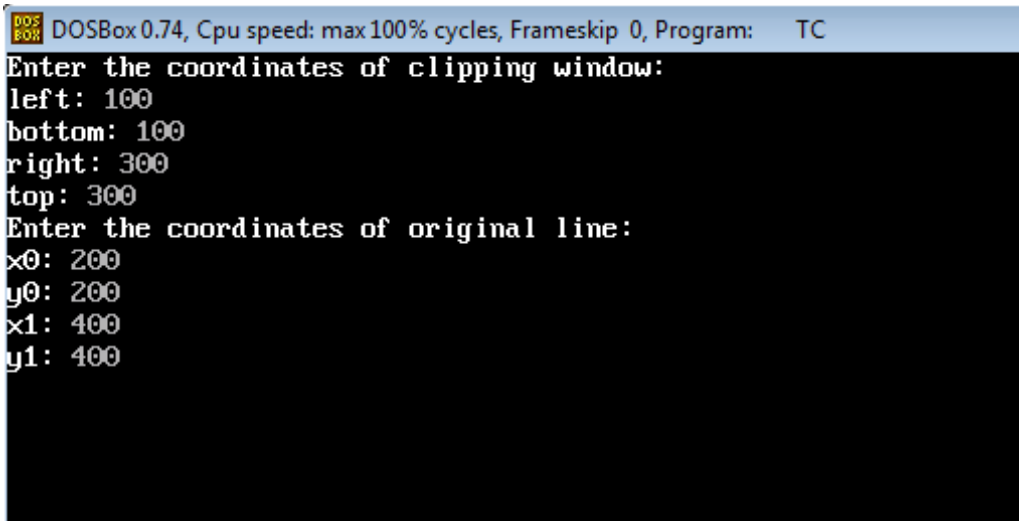
```
double xmin,double xmax,
double ymin,double ymax)
{
int outcode0,outcode1,outcodeout;
int acc=0,done=0;
outcode0=compOutCode(x0,y0,xmin,xmax,ymin,ymax);
outcode1=compOutCode(x1,y1,xmin,xmax,ymin,ymax);
do
{
 if(!(outcode0|outcode1))
 {
 acc=1;done=1;
 }
 else if(outcode0&outcode1)
 { done=1;
 }
 else{
  double x,y;
  outcodeout=outcode0?outcode0:outcode1;
  if(outcodeout&0001)
  {
  x=x0+(x1-x0)*(ymax-y0)/(y1-y0);
  y=ymax;
  cout<<"\nClipping Top!!!";
  }
```

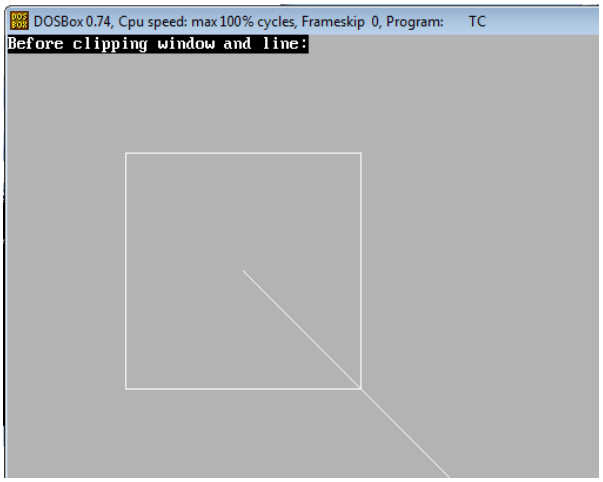
```
 else if(outcodeout&0010)
 {
 x=x0+(x1-x0)*(ymin-y0)/(y1-y0);
 y=ymin;
 cout<<"\nClipping Bottom!!!";
 }
 else if(outcodeout&0100)
 {y=y0+(y1-y0)*(xmax-x0)/(x1-x0);
 x=xmax;
 cout<<"\nClipping Right";
 }
 else
 {y=y0+(y1-y0)*(xmin-x0)/(x1-x0);
 x=xmin;
 cout<<"\nClipping Left";
 }
 if(outcodeout==outcode0)
 {
 x0=x;y0=y;outcode0=compOutCode(x0,y0,xmin,xmax,ymin,ymax);
}
else
{
x1=x;y1=y;outcode1=compOutCode(x1,y1,xmin,xmax,ymin,ymax);
}
 }
```

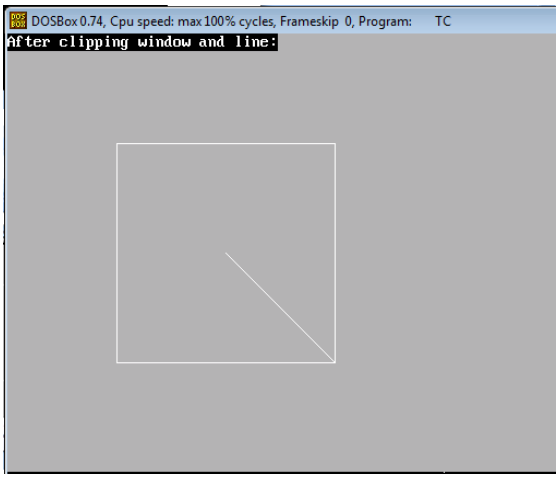
```
}
while(done==0);
clrscr();
cout<<"After clipping window and line:";
rectangle(xmin,ymax,xmax,ymin);
line(x0,y0,x1,y1);
getch();
 return;
}
void main()
{
int i,gd=DETECT,gm;
int l,r,b,t;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
cout<<"Enter the coordinates of clipping window:\n";
cout<<"left: ";
cin>>l;
cout<<"bottom: ";
cin>>b;
cout<<"right: ";
cin>>r;
cout<<"top: ";
cin>>t;
int x0,x1,y0,y1;
cout<<"Enter the coordinates of original line:\n";
```

```
cout<<"x0: ";
cin>>x0;
cout<<"y0: ";
cin>>y0;
cout<<"x1: ";
cin>>x1;
cout<<"y1: ";
cin>>y1;
clrscr();
cout<<"Before clipping window and line:";
rectangle(l,t,r,b);
line(x0,y0,x1,y1);
getch();
cohen(x0,y0,x1,y1,l,r,b,t);
 return;
}
```

OUTPUT


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter the coordinates of clipping window:
left: 100
bottom: 100
right: 300
top: 300
Enter the coordinates of original line:
x0: 200
y0: 200
x1: 400
y1: 400


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Before clipping window and line:


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
After clipping window and line:

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

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
int k,xmin,ymin,xmax,ymax,arr[20],m;
void clipl (int x1, int y1, int x2, int y2)
{
   if(x2-x1)
      m=(y2-y1)/(x2-x1);
   else
      m=100000;
   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 clipt(int x1, int y1, int x2, int y2)
{
    if(y2-y1)
        m=(x2-x1)/(y2-y1);
    else
        m=100000;
    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 clipr(int x1, int y1, int x2, int y2)
{
    if(x2-x1)
        m=(y2-y1)/(x2-x1);
    else
        m=100000;
    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 clipb(int x1, int y1, int x2, int y2)
{
    if(y2-y1)
        m=(x2-x1)/(y2-y1);
    else
        m=100000;
    if(y1 >= ymin && y2 >= ymin)
    {
        arr[k]=x2;
        arr[k+1]=y2;
        k+=2;
    }
    if(y1 < ymin && y2 >= ymin)
```

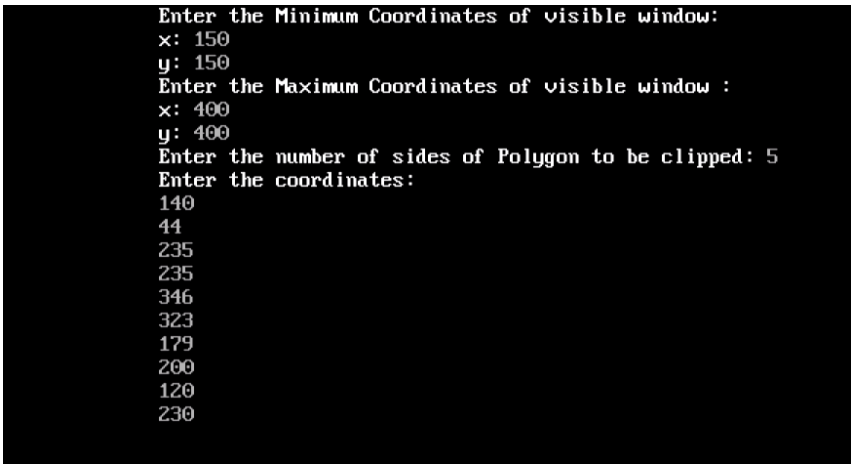
```
    {
       arr[k]=x1+m*(ymin-y1);
       arr[k+1]=ymin;
       arr[k+2]=x2;
       arr[k+3]=y2;
       k+=4;
    }
    if(y1 >= ymin  && y2 < ymin)
    {
       arr[k]=x1+m*(ymin-y1);
       arr[k+1]=ymin;
       k+=2;
    }
}
void main()
{
    int gd=DETECT,gm,n,poly[20];
    int xi,yi,xf,yf,polyy[20];
    initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
    setcolor(WHITE);
    cout<<"Enter the Minimum Coordinates of visible window:\n";
    cout<<"x: ";
    cin>>xmin;
    cout<<"y: ";
    cin>>ymin;
```

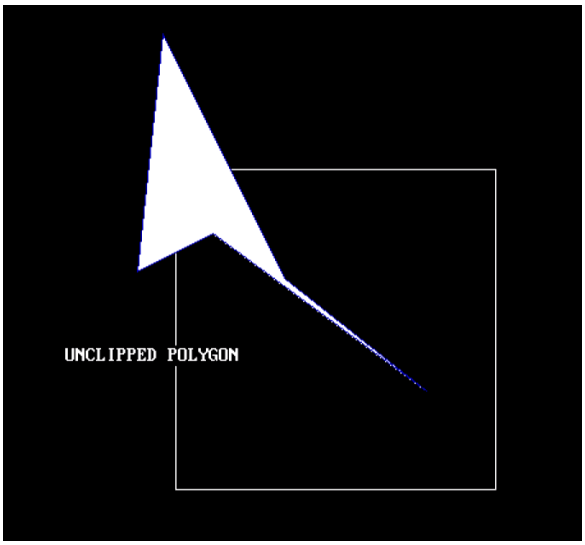
```
    cout<<"Enter the Maximum Coordinates of visible window :\n";
    cout<<"x: ";
    cin>>xmax;
    cout<<"y: ";
    cin>>ymax;
    cout<<"Enter the number of sides of Polygon to be clipped: ";
    cin>>n;
    cout<<"Enter the coordinates:\n";
    for(int i=0 ; i < 2*n ; i++)
    cin>>polyy[i];
    polyy[i]=polyy[0];
    polyy[i+1]=polyy[1];
    for(i=0 ; i < 2*n+2 ; i++)
   poly[i]=polyy[i];
    cleardevice();
    rectangle(xmin,ymax,xmax,ymin);
    cout<<"\tUNCLIPPED POLYGON";
    setcolor(BLUE);
    fillpoly(n,poly);
    getch();
    cleardevice();
    k=0;
    for(i=0;i < 2*n;i+=2)
clipl(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
    n=k/2;
```

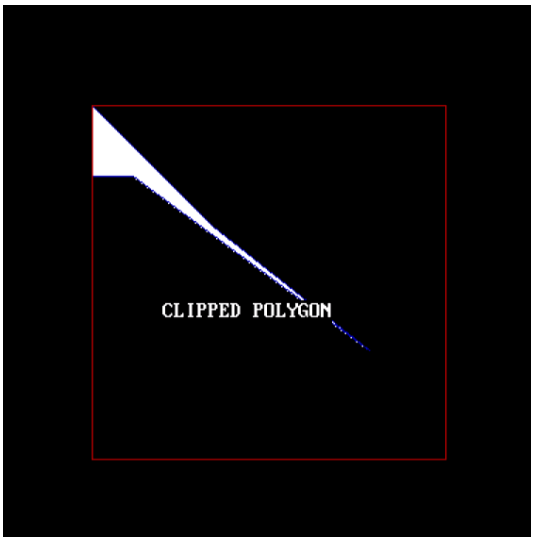
```
    for(i=0;i < k;i++)
polyy[i]=arr[i];
    polyy[i]=polyy[0];
    polyy[i+1]=polyy[1];
    k=0;
    for(i=0;i < 2*n;i+=2)
                clipt(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
    n=k/2;
    for(i=0;i < k;i++)
                polyy[i]=arr[i];
    polyy[i]=polyy[0];
    polyy[i+1]=polyy[1];
    k=0;
    for(i=0;i < 2*n;i+=2)
                clipr(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)
                clipb(polyy[i],polyy[i+1],polyy[i+2],polyy[i+3]);
    for(i=0;i < k;i++)
                poly[i]=arr[i];
```

```
    if(k)
                    fillpoly(k/2,poly);
    setcolor(RED);
    rectangle(xmin,ymax,xmax,ymin);
    cout<<"CLIPPED POLYGON";
    getch();
    closegraph();
}
```

OUTPUT







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

```
#include<conio.h>
#include<iostream.h>
#include<graphics.h>
#include<stdlib.h>
#include<dos.h>
class point{
public:
int x,y;
};
point p[20];
int inter[20],x,y;
int v,xmin,ymin,xmax,ymax;
int c;
void read()
{
cout<<"Scan Line Filling Algorithm";
cout<<"\nEnter number of vertices of polygon: ";
cin>>v;
cout<<"\nEnter the coordinates: ";
for(int i=0; i<v; i++)
{
cout<<"x"<<(i+1)<<"=";
cin>>p[i].x;
```

```
cout<<"y"<<(i+1)<<"=";
cin>>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;
}
void calcs()
{
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;
}
}
void ints(float z)
{
int x1,x2,y1,y2,temp;
```

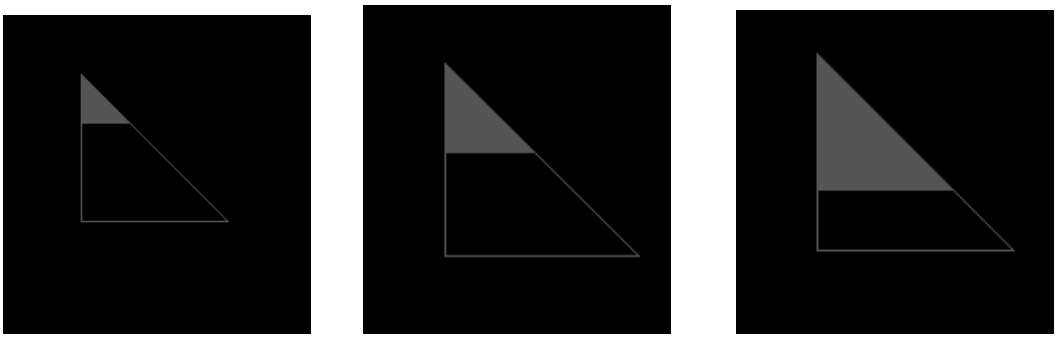
```
c=0;
for(int i=0;i<v;i++)
{
x1=p[i].x;
y1=p[i].y;
x2=p[i+1].x;
y2=p[i+1].y;
if(y2<y1)
{
temp=x1;
x1=x2;
x2=temp;
temp=y1;
y1=y2;
y2=temp;
}
if(z<=y2&&z>=y1)
{
        if((y1-y2)==0)
        x=x1;
        else
        {
        x=((x2-x1)*(z-y1))/(y2-y1);
        x=x+x1;
        }
```

```
            if(x<=xmax&&x>=xmin)
            inter[c++]=x;
            }
}
}
void sort(int z)
{
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);
}
delay(100);
for(i=0;i<c;i=i+2)
{
delay(100);
line(inter[i],z,inter[i+1],z);
}
}
void display()
{
float s,s2;
s=ymin+0.01;
cleardevice();
while(s<=ymax)
```

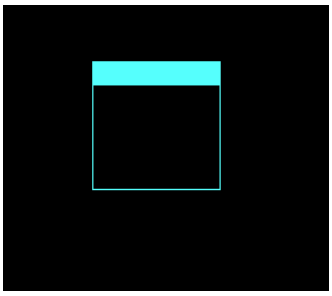
```
{
ints(s);
sort(s);
s++;
}
}
int main()
{
int gd=DETECT, gm;
initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
int cl;
cout<<"\nEnter the colour: (0-15) -> ";
cin>>cl;
setcolor(cl);
read();
calcs();
cleardevice();
display();
closegraph();
getch();
return 0;
}
```

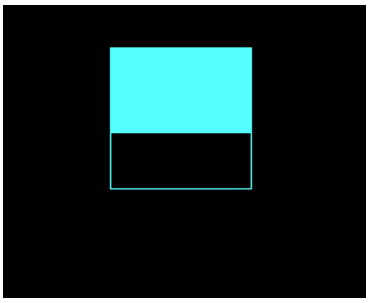
OUTPUT

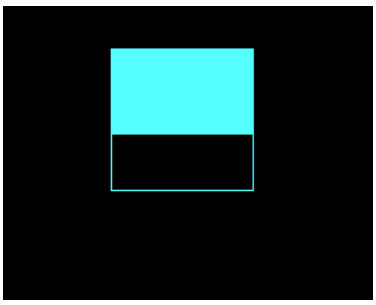
```
Enter the colour: (0-15) -> 8
Scan Line Filling Algorithm
Enter number of vertices of polygon: 3

Enter the coordinates:
x1=100
y1=100
x2=200
y2=200
x3=100
y3=200
```



```
Enter the colour: (0-15) -> 11
Scan Line Filling Algorithm
Enter number of vertices of polygon: 4

Enter the coordinates:
x1=100
y1=100
x2=100
y2=200
x3=200
y3=200
x4=200
y4=100
```







```
Enter the colour: (0-15) -> 3
Scan Line Filling Algorithm
Enter number of vertices of polygon: 10

Enter the coordinates:
x1=20
y1=50
x2=60
y2=50
x3=70
y3=20
x4=80
y4=50
x5=120
y5=50
x6=85
y6=70
x7=90
y7=120
x8=70
y8=80
x9=50
y9=120
x10=55
y10=70
```

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

```
#include<iostream.h>
#include<graphics.h>
#include<conio.h>
#include<dos.h>
#include<math.h>

#define pi 3.14285714

class transformations
{
double vertices[3][3];
double t_matrix[3][3];
double result[3][3];

public:
transformations(){};

void get_vertices();
void display_triangle();
void display_triangle_result();

void multiplication();
void copyback();

void rotation(double angle,double m,double n);
void reflection(double m,double c);
void scaling(double a,double d);
void shearing(double b,double c);

};


void transformations::get_vertices()
{
int i=0;

for(i=0;i<3;i++)
    {
        cout<<"\nEnter vertex "<<i+1<<":";
        cout<<"\nx1 : ";
        cin>>vertices[i][0];
        result[i][0]=vertices[i][0];

        cout<<"y1 : ";
```

```
        cin>>vertices[i][1];
        result[i][1]=vertices[i][1];

        vertices[i][2]=result[i][2]=1;
    }
}

void transformations::display_triangle()
{
int i=0;

for(i=0;i<2;i++)
    line(vertices[i][0],vertices[i][1],vertices[i+1][0],vertices[i+1][1]);
    line(vertices[i][0],vertices[i][1],vertices[0][0],vertices[0][1]);
}

void transformations::display_triangle_result()
{
int i=0;

for(i=0;i<2;i++)
    line(result[i][0],result[i][1],result[i+1][0],result[i+1][1]);
    line(result[i][0],result[i][1],result[0][0],result[0][1]);
}

void transformations::copyback()
{
int i=0,j=0;

for(i=0;i<3;i++)
for(j=0;j<3;j++)
result[i][j]=vertices[i][j];
}

void transformations::multiplication()
{
double r[3][3];
int i=0,j=0,k=0;

        for(i=0;i<3;i++)
         {
        for(j=0;j<3;j++)
            {
                r[i][j]=0;
                for(k=0;k<3;k++)
                r[i][j]+=result[i][k]*t_matrix[k][j];
            }
         }
```

```
for(i=0;i<3;i++)
for(j=0;j<3;j++)
result[i][j]=r[i][j];
}


void transformations::rotation(double angle,double m,double n)
{
angle=((pi/180)*angle);

copyback();
cleardevice();

setcolor(RED);
display_triangle();
delay(20);
getch();

t_matrix[0][0]=1;
t_matrix[0][1]=0;
t_matrix[0][2]=0;
t_matrix[1][0]=0;
t_matrix[1][1]=1;
t_matrix[1][2]=0;
t_matrix[2][0]=(m*(-1));
t_matrix[2][1]=(n*(-1));
t_matrix[2][2]=1;

        multiplication();

t_matrix[0][0]=cos(angle);
t_matrix[0][1]=sin(angle);
t_matrix[1][0]=(sin(angle)*(-1));;
t_matrix[1][1]=cos(angle);
t_matrix[2][0]=0;
t_matrix[2][1]=0;

        multiplication();

t_matrix[0][0]=1;
t_matrix[0][1]=0;
t_matrix[1][0]=0;
t_matrix[1][1]=1;
t_matrix[2][0]=m;
t_matrix[2][1]=n;

        multiplication();
```

```
setcolor(BLUE);
display_triangle_result();
delay(20);
getch();
}

void transformations::reflection(double m,double c)
{
double angle=atan(m);

copyback();
cleardevice();

double x1=0,y1=c,x2=400,y2=(m*x2)+c;
setcolor(YELLOW);
line(x1,y1,x2,y2);
delay(20);
getch();

setcolor(RED);
display_triangle();
delay(20);
getch();

t_matrix[0][0]=1;
t_matrix[0][1]=0;
t_matrix[0][2]=0;
t_matrix[1][0]=0;
t_matrix[1][1]=1;
t_matrix[1][2]=0;
t_matrix[2][0]=0;
t_matrix[2][1]=(c*(-1));
t_matrix[2][2]=1;

        multiplication();

t_matrix[0][0]=cos(-1*angle);
t_matrix[0][1]=sin(-1*angle);
t_matrix[1][0]=(sin(-1*angle)*(-1));;
t_matrix[1][1]=cos(-1*angle);
t_matrix[2][0]=0;
t_matrix[2][1]=0;

        multiplication();

t_matrix[0][0]=1;
t_matrix[0][1]=0;
```

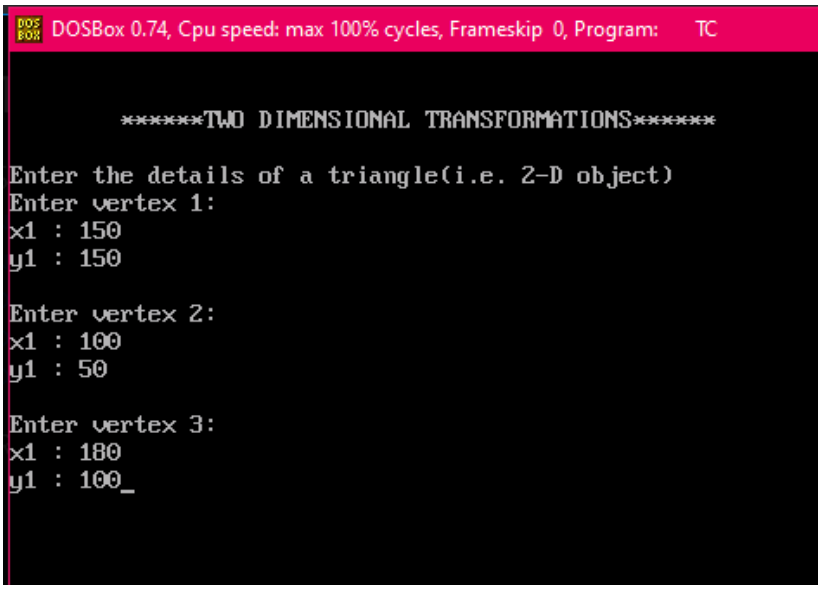
```
t_matrix[1][0]=0;
t_matrix[1][1]=-1;
t_matrix[2][0]=0;
t_matrix[2][1]=0;

        multiplication();

t_matrix[0][0]=cos(angle);
t_matrix[0][1]=sin(angle);
t_matrix[1][0]=(sin(angle)*(-1));;
t_matrix[1][1]=cos(angle);
t_matrix[2][0]=0;
t_matrix[2][1]=0;

        multiplication();

t_matrix[0][0]=1;
t_matrix[0][1]=0;
t_matrix[1][0]=0;
t_matrix[1][1]=1;
t_matrix[2][0]=0;
t_matrix[2][1]=c;

        multiplication();

setcolor(BLUE);
display_triangle_result();
delay(20);
getch();
}


void transformations::scaling(double a,double d)
{
copyback();
cleardevice();

setcolor(RED);
display_triangle();
delay(20);
getch();

t_matrix[0][0]=a;
t_matrix[0][1]=0;
t_matrix[0][2]=0;
t_matrix[1][0]=0;
t_matrix[1][1]=d;
t_matrix[1][2]=0;
```

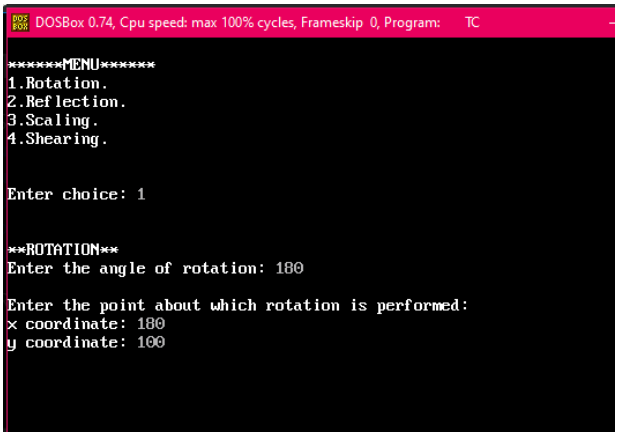
```
t_matrix[2][0]=0;
t_matrix[2][1]=0;
t_matrix[2][2]=1;

        multiplication();

setcolor(BLUE);
display_triangle_result();
delay(20);
getch();
}


void transformations::shearing(double b,double c)
{
copyback();
cleardevice();

setcolor(RED);
display_triangle();
delay(20);
getch();

t_matrix[0][0]=1;
t_matrix[0][1]=b;
t_matrix[0][2]=0;
t_matrix[1][0]=c;
t_matrix[1][1]=1;
t_matrix[1][2]=0;
t_matrix[2][0]=0;
t_matrix[2][1]=0;
t_matrix[2][2]=1;

        multiplication();

setcolor(BLUE);
display_triangle_result();
delay(20);
getch();
}


void main()
{
clrscr();
int gd=DETECT,gm,choice;
transformations t1;
char ch1,ch2;
```

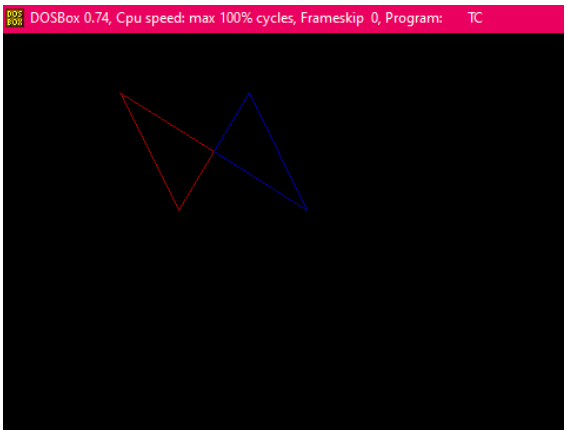
```
double angle,m,n,slope,intercept,a,b,c,d;

do
  {
cout<<"\n\n\t******TWO DIMENSIONAL TRANSFORMATIONS******\n";
cout<<"\nEnter the details of a triangle(i.e. 2-D object)";
    t1.get_vertices();

        do
         {
        initgraph(&gd,&gm,"C:\\Turboc3\\BGI");
        cout<<"\n******MENU******";
        cout<<"\n1.Rotation.";
        cout<<"\n2.Reflection.";
        cout<<"\n3.Scaling.";
        cout<<"\n4.Shearing.";
        cout<<"\n";

        cout<<"\n\nEnter choice: ";
        cin>>choice;

        switch(choice)
            {
                case 1:cout<<"\n\n**ROTATION**";
                        cout<<"\nEnter the angle of rotation: ";
                        cin>>angle;

                        cout<<"\nEnter the point about which rotation is performed: ";
                        cout<<"\nx coordinate: ";
                        cin>>m;
                        cout<<"y coordinate: ";
                        cin>>n;

                        t1.rotation(angle,m,n);
                        break;

                case 2:cout<<"\n\n**REFLECTION**";
                        cout<<"\nTo enter the line in slope-intercept form(i.e. y=mx+b)";
                        cout<<"\nEnter slope(m): ";
                        cin>>slope;
                        cout<<"Enter y-intercept(b): ";
                        cin>>intercept;

                        t1.reflection(slope,intercept);
                        break;

                case 3:cout<<"\n\n**SCALING**";
                        cout<<"\nEnter the factor of scaling";
```

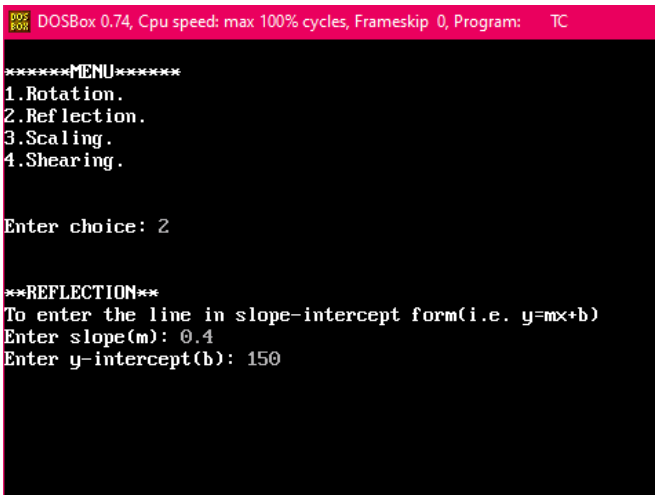
```
                cout<<"\nAlong the x-axis: ";
                cin>>a;
                cout<<"Along the y-axis: ";
                cin>>d;

                t1.scaling(a,d);
                break;

        case 4:cout<<"\n\n**SHEARING**";
                cout<<"\nEnter the factor of shearing";
                cout<<"\nAlong the x-axis: ";
                cin>>c;
                cout<<"Along the y-axis: ";
                cin>>b;

                t1.shearing(b,c);
                break;

        default:cout<<"\n\n\tINVALID CHOICE!!!";
                getch();
    }

    closegraph();

    cout<<"\nDo you want to try another transformation?(Y/N): ";
    cin>>ch2;

    }while(ch2=='y'|| ch2=='Y');

cout<<"\n\nDo you want to try with a triangle with different dimensions(Y/N)? ";
cin>>ch1;
}while(ch1=='y' || ch1=='Y');
}
```

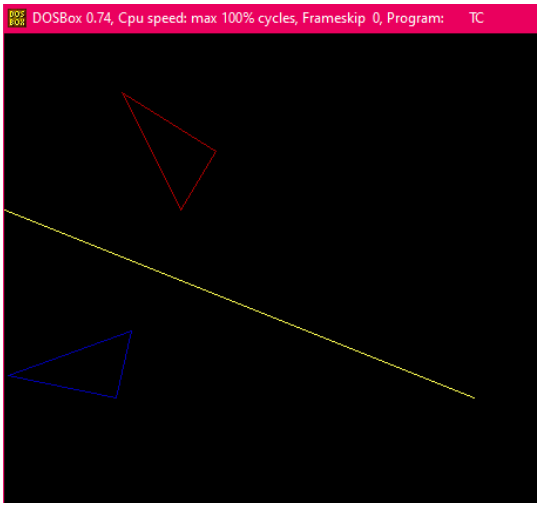
OUTPUT

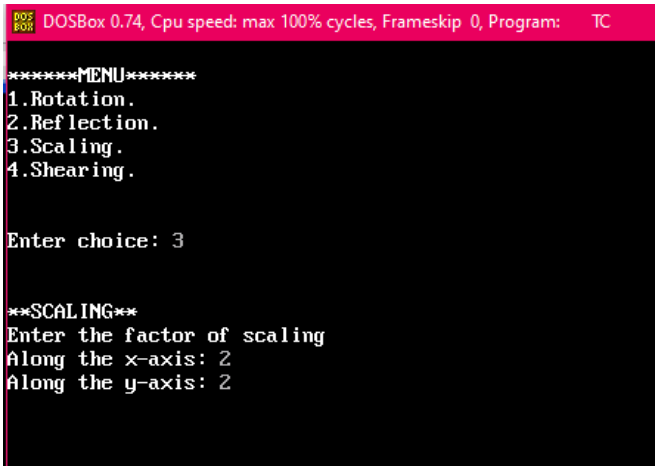

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
******TWO DIMENSIONAL TRANSFORMATIONS******
Enter the details of a triangle(i.e. 2-D object)
Enter vertex 1:
x1 : 150
y1 : 150
Enter vertex 2:
x1 : 100
y1 : 50
Enter vertex 3:
x1 : 180
y1 : 100

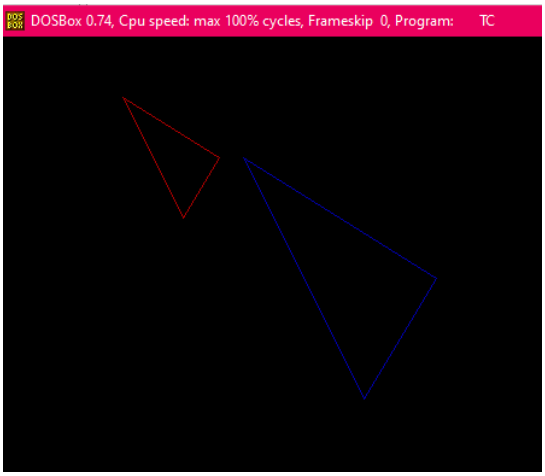

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
******MENU******
1.Rotation.
2.Reflection.
3.Scaling.
4.Shearing.
Enter choice: 1
ROTATION
Enter the angle of rotation: 180
Enter the point about which rotation is performed:
x coordinate: 180
y coordinate: 100

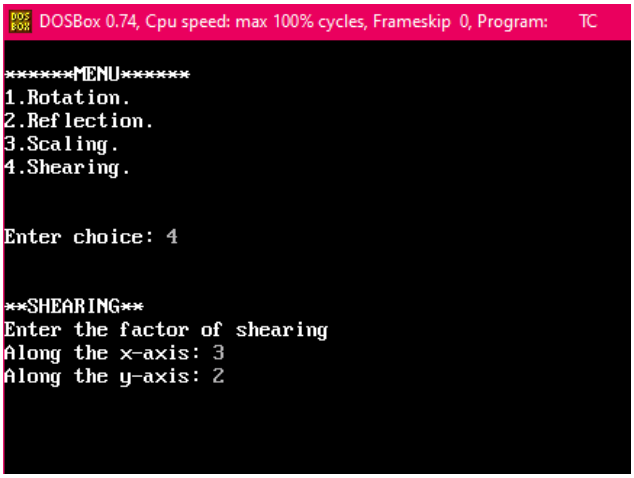

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

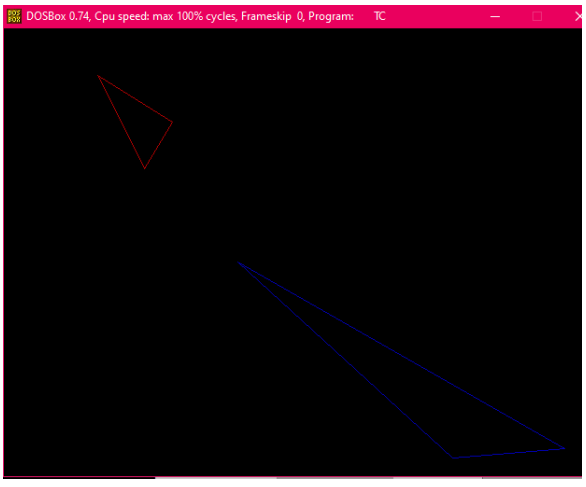

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
******MENU******
1.Rotation.
2.Reflection.
3.Scaling.
4.Shearing.
Enter choice: 2
REFLECTION
To enter the line in slope-intercept form(i.e. y=mx+b)
Enter slope(m): 0.4
Enter y-intercept(b): 150


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
******MENU******
1.Rotation.
2.Reflection.
3.Scaling.
4.Shearing.
Enter choice: 3
SCALING
Enter the factor of scaling
Along the x-axis: 2
Along the y-axis: 2


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
******MENU******
1.Rotation.
2.Reflection.
3.Scaling.
4.Shearing.
Enter choice: 4
SHEARING
Enter the factor of shearing
Along the x-axis: 3
Along the y-axis: 2


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

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

```
#include<iostream.h>
#include<graphics.h>
#include<conio.h>
#include<dos.h>
#include<math.h>

#define pi 3.14285714

class projections
{
    double vertices[8][4];
    double t_matrix[4][4];
    double result[8][4];

    public:
    projections(){};

    void get_vertices();
    void display_cube();

    void multiplication();
    void copyback();

    void orthographic();
    void axonometric(double angle_x,double angle_y);
    void cavalier(double angle);
    void cabinet(double angle);
    void single_point(double r);
    void two_point(double r);
    void three_point(double r);
```

```
};


void projections::get_vertices()
{
    for(int i=0;i<8;i++)
    {
            cout<<"\nEnter vertex "<<i+1<<": ";
            cout<<"\nx: ";
            cin>>vertices[i][0];
            result[i][0]=vertices[i][0];

            cout<<"y: ";
            cin>>vertices[i][1];
            result[i][1]=vertices[i][1];

            cout<<"z: ";
            cin>>vertices[i][2];
            result[i][2]=vertices[i][2];

            vertices[i][3]=result[i][3]=1;
    }
}

void projections::display_cube()
{
    int i=0;

    for(i=0;i<3;i++)
    line(result[i][0],result[i][1],result[i+1][0],result[i+1][1]);
    line(result[i][0],result[i][1],result[0][0],result[0][1]);

    for(i=4;i<7;i++)
    line(result[i][0],result[i][1],result[i+1][0],result[i+1][1]);
    line(result[i][0],result[i][1],result[4][0],result[4][1]);

    for(i=0;i<4;i++)
    line(result[i][0],result[i][1],result[i+4][0],result[i+4][1]);
}

void projections::copyback()
{
    int i=0,j=0;

    for(i=0;i<8;i++)
    for(j=0;j<4;j++)
    result[i][j]=vertices[i][j];
}
```

```
void projections::multiplication()
{
    double r[8][4];
    int i=0,j=0,k=0;

            for(i=0;i<8;i++)
            {
               for(j=0;j<4;j++)
               {
                        r[i][j]=0;
                        for(k=0;k<4;k++)
                        r[i][j]+=result[i][k]*t_matrix[k][j];
               }
            }

    for(i=0;i<8;i++)
    for(j=0;j<4;j++)
    result[i][j]=r[i][j];
}

void projections::orthographic()
{
    cleardevice();
    clearviewport();
    cout<<"\t**ORTHOGRAPHIC PROJECTION**";

    copyback();

    t_matrix[0][0]=1;
    t_matrix[0][1]=0;
    t_matrix[0][2]=0;
    t_matrix[0][3]=0;

    t_matrix[1][0]=0;
    t_matrix[1][1]=1;
    t_matrix[1][2]=0;
    t_matrix[1][3]=0;

    t_matrix[2][0]=0;
    t_matrix[2][1]=0;
    t_matrix[2][2]=0;
    t_matrix[2][3]=0;

    t_matrix[3][0]=0;
    t_matrix[3][1]=0;
    t_matrix[3][2]=0;
    t_matrix[3][3]=1;
```

```
                multiplication();

   setcolor(BLUE);
   display_cube();
   delay(20);
   getch();
}

void projections::axonometric(double angle_x,double angle_y)
{
    angle_x=((pi/180)*angle_x);
    angle_y=((pi/180)*angle_y);
    cleardevice();
    clearviewport();
    cout<<"\t**AXONOMETRIC PROJECTION**";

    copyback();

    t_matrix[0][0]=cos(angle_y);
    t_matrix[0][1]=(sin(angle_y))*(sin(angle_x));
    t_matrix[0][2]=0;
    t_matrix[0][3]=0;

    t_matrix[1][0]=0;
    t_matrix[1][1]=cos(angle_x);
    t_matrix[1][2]=0;
    t_matrix[1][3]=0;

    t_matrix[2][0]=sin(angle_y);
    t_matrix[2][1]=(-1)*(cos(angle_y)*sin(angle_x));
    t_matrix[2][2]=0;
    t_matrix[2][3]=0;

    t_matrix[3][0]=0;
    t_matrix[3][1]=0;
    t_matrix[3][2]=0;
    t_matrix[3][3]=1;

                multiplication();

   setcolor(BLUE);
   display_cube();
   delay(20);
   getch();
}

void projections::cavalier(double angle)
```

```
{
    angle=((pi/180)*angle);

    cleardevice();
    clearviewport();
    cout<<"\t**CAVALIER PROJECTION**";

    copyback();

    t_matrix[0][0]=1;
    t_matrix[0][1]=0;
    t_matrix[0][2]=0;
    t_matrix[0][3]=0;

    t_matrix[1][0]=0;
    t_matrix[1][1]=1;
    t_matrix[1][2]=0;
    t_matrix[1][3]=0;

    t_matrix[2][0]=(-1)*1*cos(angle);
    t_matrix[2][1]=(-1)*1*sin(angle);
    t_matrix[2][2]=0;
    t_matrix[2][3]=0;

    t_matrix[3][0]=0;
    t_matrix[3][1]=0;
    t_matrix[3][2]=0;
    t_matrix[3][3]=1;

                multiplication();

   setcolor(BLUE);
   display_cube();
   delay(20);
   getch();
}

void projections::cabinet(double angle)
{
    angle=((pi/180)*angle);

    cleardevice();
    clearviewport();
    cout<<"\t**CABINET PROJECTION**";

    copyback();

    t_matrix[0][0]=1;
```

```
    t_matrix[0][1]=0;
    t_matrix[0][2]=0;
    t_matrix[0][3]=0;

    t_matrix[1][0]=0;
    t_matrix[1][1]=1;
    t_matrix[1][2]=0;
    t_matrix[1][3]=0;

    t_matrix[2][0]=(-1)*0.5*cos(angle);
    t_matrix[2][1]=(-1)*0.5*sin(angle);
    t_matrix[2][2]=0;
    t_matrix[2][3]=0;

    t_matrix[3][0]=0;
    t_matrix[3][1]=0;
    t_matrix[3][2]=0;
    t_matrix[3][3]=1;

                multiplication();

   setcolor(BLUE);
   display_cube();
   delay(20);
   getch();
}

void projections::single_point(double r)
{
    double l=10,m=10,n=10;

    cleardevice();
    clearviewport();
    cout<<"\t**SINGLE POINT PRESPECTIVE PROJECTION**";

    r=(-1/r);
    copyback();

    t_matrix[0][0]=1;
    t_matrix[0][1]=0;
    t_matrix[0][2]=0;
    t_matrix[0][3]=0;

    t_matrix[1][0]=0;
    t_matrix[1][1]=1;
    t_matrix[1][2]=0;
    t_matrix[1][3]=0;
```

```
    t_matrix[2][0]=0;
    t_matrix[2][1]=0;
    t_matrix[2][2]=0;
    t_matrix[2][3]=r;

    t_matrix[3][0]=l;
    t_matrix[3][1]=m;
    t_matrix[3][2]=0;
    t_matrix[3][3]=r*n+1;

    multiplication();

    setcolor(BLUE);
    display_cube();
    delay(20);
    getch();
}

void projections::two_point(double r)
{
    double angle=45;
    angle=(pi/180)*angle;

    cleardevice();
    clearviewport();
    cout<<"\t**TWO POINT PRESPECTIVE PROJECTION**";

    copyback();

    r=(-1/r);

    t_matrix[0][0]=cos(angle);
    t_matrix[0][1]=0;
    t_matrix[0][2]=(-1*sin(angle));
    t_matrix[0][3]=sin(angle)/r;

    t_matrix[1][0]=0;
    t_matrix[1][1]=1;
    t_matrix[1][2]=0;
    t_matrix[1][3]=0;

    t_matrix[2][0]=sin(angle);
    t_matrix[2][1]=0;
    t_matrix[2][2]=cos(angle);
    t_matrix[2][3]=(-1*cos(angle))/r;

    t_matrix[3][0]=0;
    t_matrix[3][1]=0;
```

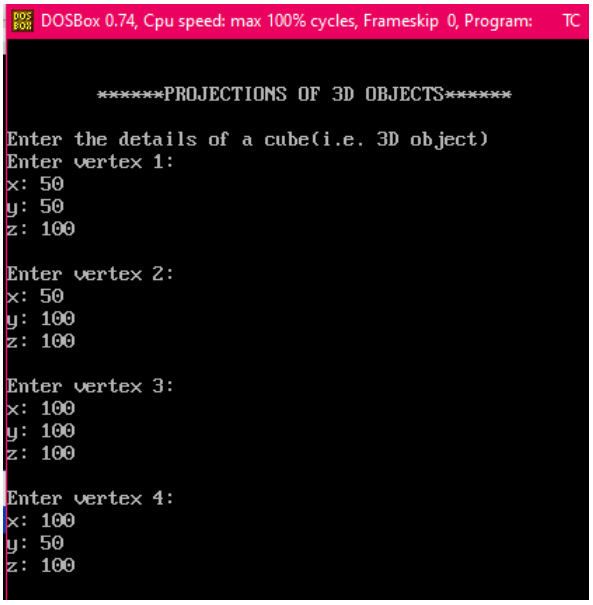
```
    t_matrix[3][2]=0;
    t_matrix[3][3]=1;

                multiplication();

    setcolor(BLUE);
    display_cube();
    delay(20);
    getch();
}

void projections::three_point(double r)
{
    double angle_y=45,angle_x=45;
    angle_y=(pi/180)*angle_y;
    angle_x=(pi/180)*angle_x;

    cleardevice();
    clearviewport();
    cout<<"\t**THREE POINT PRESPECTIVE PROJECTION**";

    copyback();

    r=(-1/r);

    t_matrix[0][0]=cos(angle_y);
    t_matrix[0][1]=sin(angle_y)*sin(angle_x);
    t_matrix[0][2]=0;
    t_matrix[0][3]=(sin(angle_y)*cos(angle_x))/r;

    t_matrix[1][0]=0;
    t_matrix[1][1]=cos(angle_x);
    t_matrix[1][2]=0;
    t_matrix[1][3]=(-1*sin(angle_x))/r;

    t_matrix[2][0]=(sin(angle_y));
    t_matrix[2][1]=(-1*cos(angle_y)*sin(angle_x));
    t_matrix[2][2]=0;
    t_matrix[2][3]=(-1*cos(angle_y)*cos(angle_x))/r;

    t_matrix[3][0]=0;
    t_matrix[3][1]=0;
    t_matrix[3][2]=0;
    t_matrix[3][3]=1;

    multiplication();

    setcolor(BLUE);
```

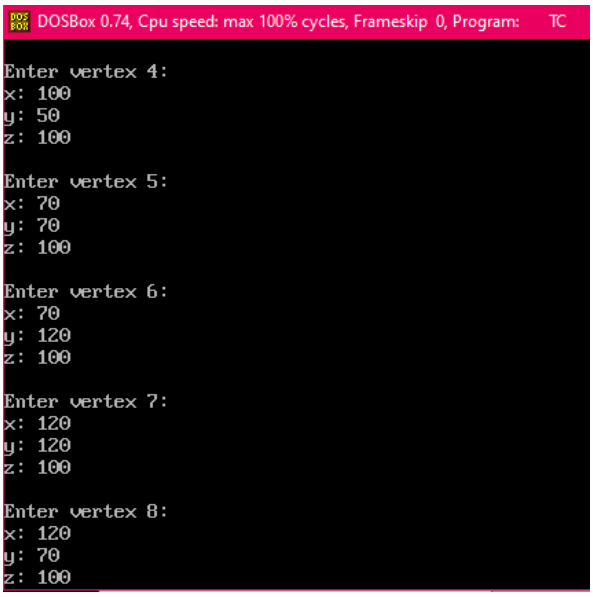
```
    display_cube();
    delay(20);
    getch();
}

void main()
{
   clrscr();
   int gd=DETECT,gm,choice;
   projections t1;
   char ch1,ch2,axis,axis1,axis2;
   double angle_x,angle_y,angle,ratio,ratio1,ratio2,ratio3;

   do
   {
     cout<<"\n\n\t******PROJECTIONS OF 3D OBJECTS******\n";
     cout<<"\nEnter the details of a cube(i.e. 3D object)";
     t1.get_vertices();

           do
           {
              initgraph(&gd,&gm,"C:\\Turboc3\\BGI");
              cout<<"\n\n******MENU******";
              cout<<"\n1.Orthographic.";
              cout<<"\n2.Axonometric.";
              cout<<"\n3.Cavalier (Oblique type 1).";
              cout<<"\n4.Cabinet (Oblique type 2)";
              cout<<"\n5.Single-Point presepective.";
              cout<<"\n6.Two-Point presepective.";
              cout<<"\n7.Three-Point presepective.";
              cout<<"\n";

              cout<<"\n\nEnter choice: ";
              cin>>choice;

              switch(choice)
              {
                      case 1:t1.orthographic();
                              break;

                      case 2:cout<<"\n\n**AXONOMETRIC PROJECTION**";
                              cout<<"\nEnter the angle of rotation about: ";
                              cout<<"\nx-axis: ";
                              cin>>angle_x;
                              cout<<"y-axis: ";
                              cin>>angle_y;

                              t1.axonometric(angle_x,angle_y);
```

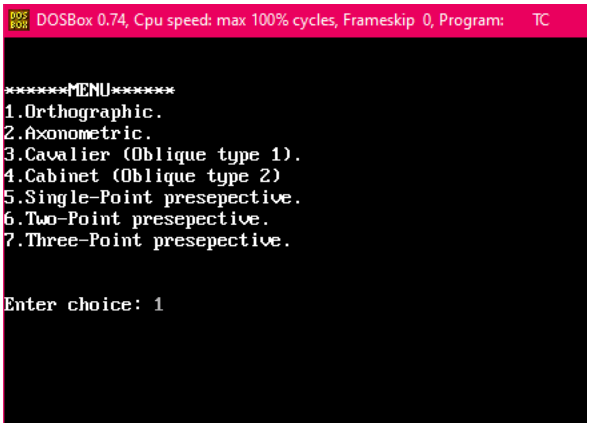
```
                        break;

                case 3:cout<<"\n\n**CAVALIER PROJECTION**";
                        cout<<"\nEnter the angle of inclination: ";
                        cin>>angle;

                        t1.cavalier(angle);
                        break;

                case 4:cout<<"\n\n**CABINET PROJECTION**";
                        cout<<"\nEnter the angle of inclination: ";
                        cin>>angle;

                        t1.cabinet(angle);
                        break;

                case 5:cout<<"\n\n**SINGLE POINT PRESPECTIVE PROJECTION**";
                        cout<<"\nAssuming that the VP lies on the z-axis, Enter the prespective ratio:
";
                        cin>>ratio;

                        t1.single_point(ratio);

                        break;

                case 6:cout<<"\n\nF**TWO POINT PRESPECTIVE PROJECTION**";
                        cout<<"\nAssuming that the VP lies on the z-axis, Enter the prespective ratio:
";
                        cin>>ratio;

                        t1.two_point(ratio);

                        break;

                case 7:cout<<"\n\n**THREE POINT PRESPECTIVE PROJECTION**";
                        cout<<"\nAssuming that the VP lies on the z-axis, Enter the prespective ratio:
";
                        cin>>ratio;

                        t1.three_point(ratio);

                        break;

                default:cout<<"\n\n\tINVALID CHOICE!!!";
                        getch();
        }
        closegraph();
```

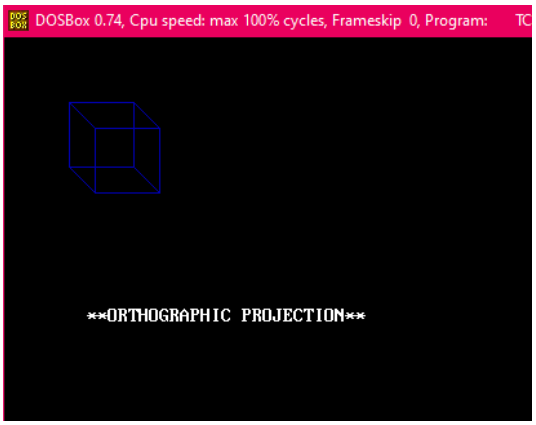
```
            cout<<"\nDo you want to try another projection(Y/N)? ";
            cin>>ch2;

        }while(ch2=='y' || ch2=='Y');

    cout<<"\nDo you want to try a cube with different dimensions(Y/N)?";
    cin>>ch1;

  }while(ch1=='y' || ch1=='Y');
}
```

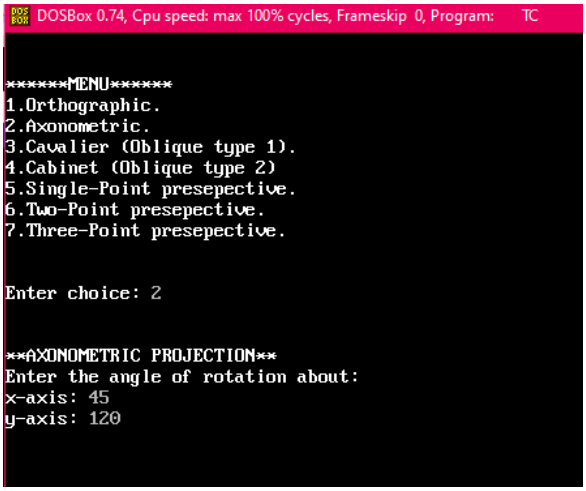
OUTPUT

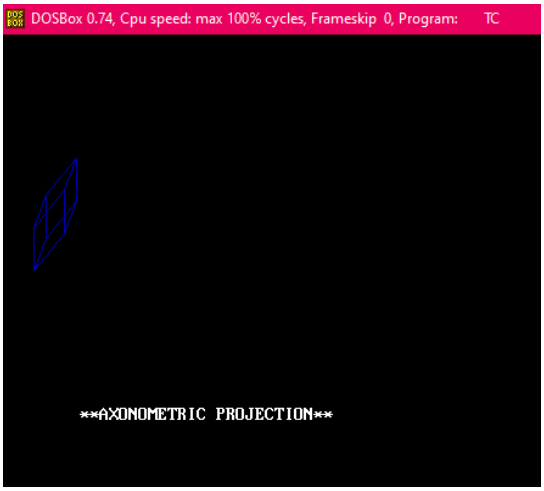


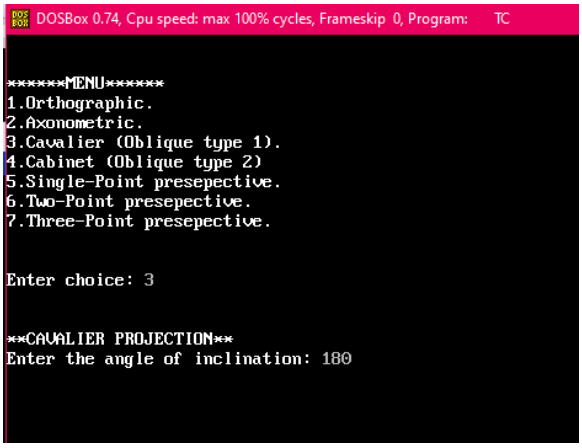


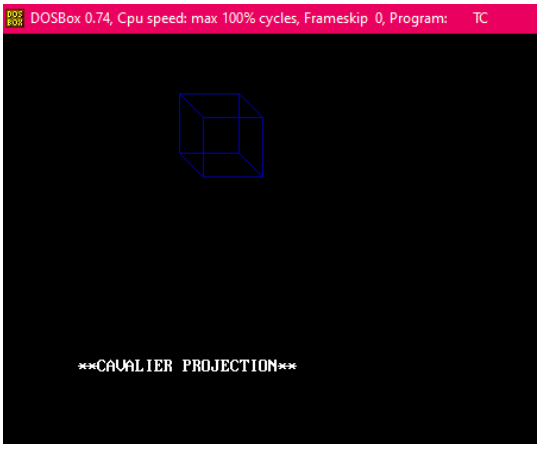


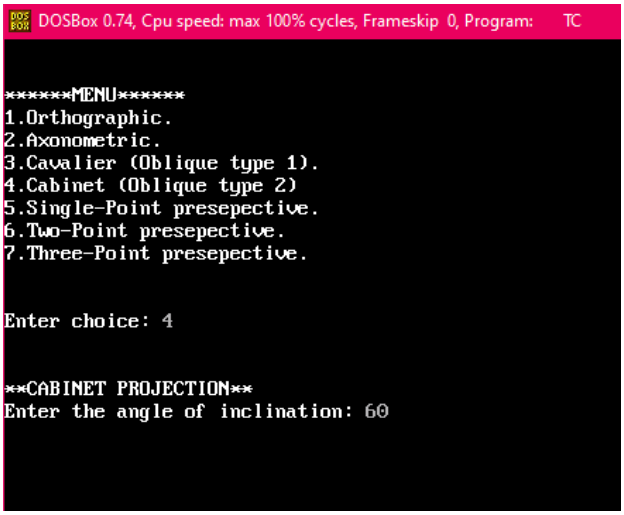


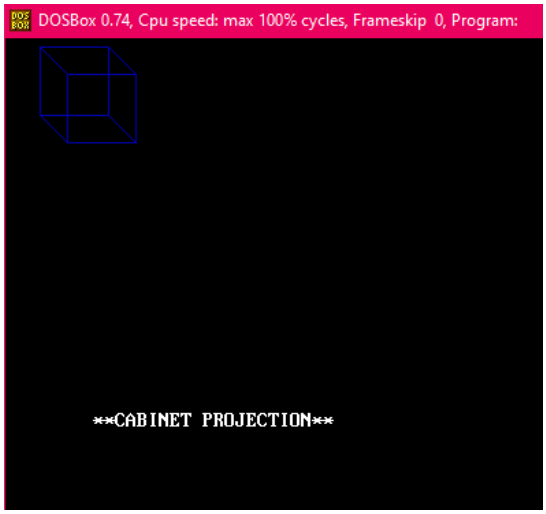

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
******MENU******
1.Orthographic.
2.Axonometric.
3.Cavalier (Oblique type 1).
4.Cabinet (Oblique type 2)
5.Single-Point presepective.
6.Two-Point presepective.
7.Three-Point presepective.
Enter choice: 2
AXONOMETRIC PROJECTION
Enter the angle of rotation about:
x-axis: 45
y-axis: 120

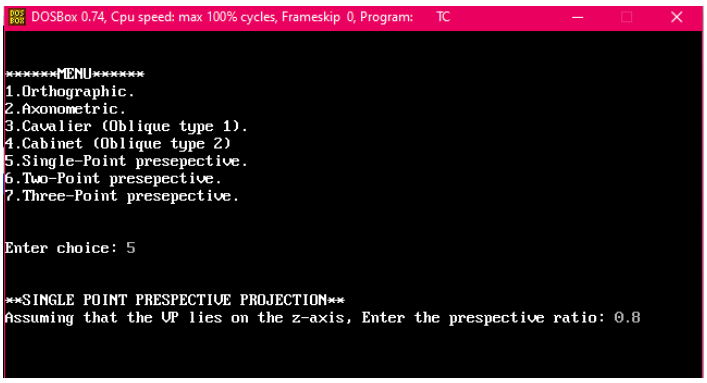

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
AXONOMETRIC PROJECTION

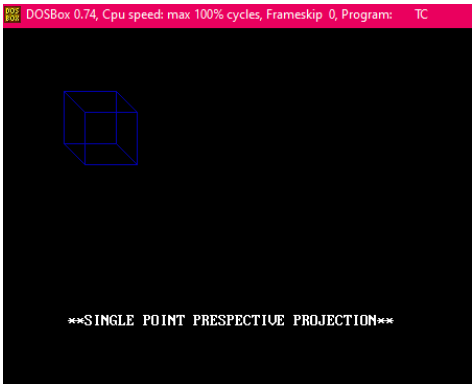

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
******MENU******
1.Orthographic.
2.Axonometric.
3.Cavalier (Oblique type 1).
4.Cabinet (Oblique type 2)
5.Single-Point presepective.
6.Two-Point presepective.
7.Three-Point presepective.
Enter choice: 3
CAVALIER PROJECTION
Enter the angle of inclination: 180

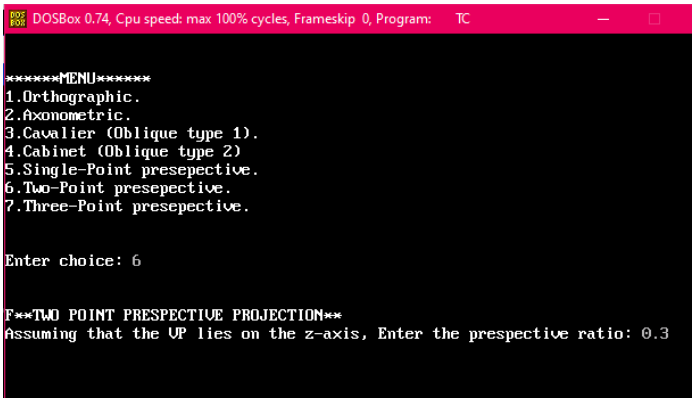

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
CAVALIER PROJECTION

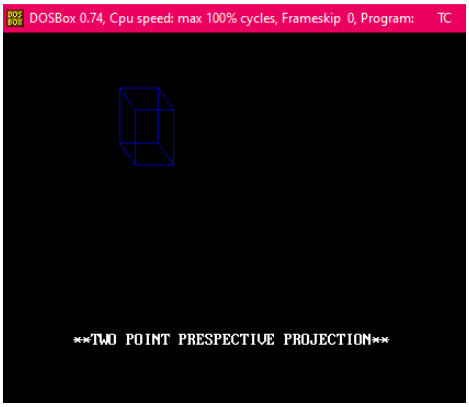

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
******MENU******
1.Orthographic.
2.Axonometric.
3.Cavalier (Oblique type 1).
4.Cabinet (Oblique type 2)
5.Single-Point presepective.
6.Two-Point presepective.
7.Three-Point presepective.
Enter choice: 4
CABINET PROJECTION
Enter the angle of inclination: 60

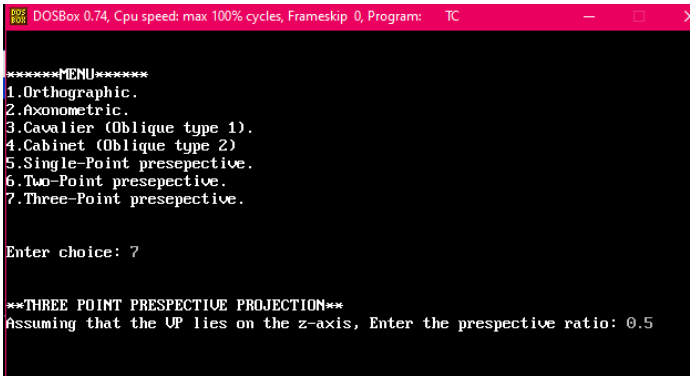

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program:
CABINET PROJECTION

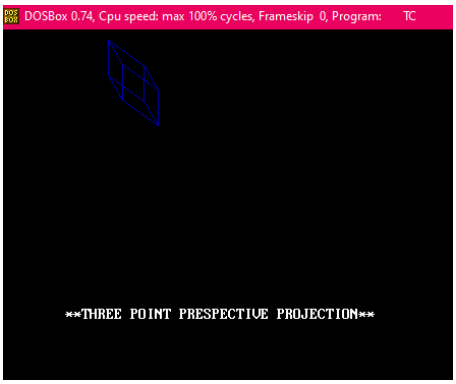

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
******MENU******
1.Orthographic.
2.Axonometric.
3.Cavalier (Oblique type 1).
4.Cabinet (Oblique type 2)
5.Single-Point presepective.
6.Two-Point presepective.
7.Three-Point presepective.
Enter choice: 5
SINGLE POINT PRESPECTIVE PROJECTION
Assuming that the VP lies on the z-axis, Enter the prespective ratio: 0.8


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
SINGLE POINT PRESPECTIVE PROJECTION


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
******MENU******
1.Orthographic.
2.Axonometric.
3.Cavalier (Oblique type 1).
4.Cabinet (Oblique type 2)
5.Single-Point presepective.
6.Two-Point presepective.
7.Three-Point presepective.
Enter choice: 6
F**TWO POINT PRESPECTIVE PROJECTION**
Assuming that the VP lies on the z-axis, Enter the prespective ratio: 0.3


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
TWO POINT PRESPECTIVE PROJECTION


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
******MENU******
1.Orthographic.
2.Axonometric.
3.Cavalier (Oblique type 1).
4.Cabinet (Oblique type 2)
5.Single-Point presepective.
6.Two-Point presepective.
7.Three-Point presepective.
Enter choice: 7
THREE POINT PRESPECTIVE PROJECTION
Assuming that the VP lies on the z-axis, Enter the prespective ratio: 0.5


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
THREE POINT PRESPECTIVE PROJECTION

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

```
#include<stdio.h>
#include<graphics.h>
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
void bezier(int x[4], int y[4])
{
   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,BLUE);
   }
   for(int i=0;i < 4;i++)
   putpixel(x[i],y[i],YELLOW);
   getch();
   closegraph();
   return;
}
void main()
```

```
{
   /* request auto detection */
   int gdriver = DETECT, gmode, errorcode;

   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "..\\bgi");

   /* read result of initialization */
   errorcode = graphresult();

   /* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }
   int x[4],y[4];
   int i;
        cout<<"Enter x and y coordinates: "<<endl;

        for(i=0;i < 4;i++)
        {
           cin>>x[i];
```

```
        cout<<endl;
        cin>>y[i];
    }
    clrscr();
    bezier(x,y);
}
```

OUTPUT