THADOMAL SHAHANI ENGINEERING COLLEGE
BANDRA (W), MUMBAI – 400050

CERTIFICATE

This is to certify that Mr. **Kasliwal Harsh Nitin** of **COMPUTER** Department, Semester **|||** With Roll No. **2003085** has completed a course of the necessary experiments in the subject "**COMPUTER GRAPHICS"** under my supervision in the **THADOMAL SHAHANI ENGINEERING COLLEGE** Laboratory in the year 2021- 2022

Prof. Manisha Dumbre

Teacher in-charge **Head of Department**

Date: _____ **Principal**

Thadomal Shahani Engineering College

Computer Engineering Department

E-JOURNAL

Subject: CG Lab Year: 2021-2022

Semester: III Class: C2

Name of the Student: Kasliwal Harsh Nitin **Batch:C21**

Roll Number: 2003085

Experiment List

SR.No.	TOPICS	DATE
1.	Implement DDA Line drawing Method in C	08/09/2021
2.	Implement Bresenham's Line Drawing Method in C	15/09/2021
3.	Implement Midpoint circle drawing Method in C	22/09/2021
4.	Implement Midpoint ellipse drawing Method in C	06/10/2021
5.	Implement flood fill and boundary fill to fill a polygon.	29/09/2021
6.	Implement 2D Transformations on a polygon – Translation, Rotation, Scaling, Reflection and Shear	20/10/2021 28/10/2021
7.	Implement Liang Barsky Line Clipping Method in C	24/11/2021
8.	Implement Sutherland Hodgeman polygon clipping Method in C	24/11/2021
9.	Implement Bezier curve	10/11/2021
10.	Implement Koch Curve for fractal generation	17/11/2021
11.	Mini Project – **(MENTION TITLE AND GROUP ROLL NUMBERS)**	08/12/2021
12.	Written Assignment No.1	01/09/2021
13.	Written Assignment No.2	10/11/2021

Prof. Manisha Dumbre

Harsh Kashiwal
2003085
C21

Page No.
Date

Experiment 1

Aim: DDA line drawing algorithm implementation.

Algorithm:

Step 0: Two end points
Step 1: Input the coordinates of two ends A(X_1, Y_1) and B(X_2, Y_2) for the line AB resp.

Step 2: Calculate
$dx = X_2 - X_1$
$dy = Y_2 - Y_1$

Step 3: Calculate the length L
if (abs($X_2 - X_1$) >= abs($Y_2 - Y_1$))
L = abs($X_2 - X_1$);
else
L = abs($Y_2 - Y_1$);

Step 4: Calculate increment factor.

→ $X_i = dx / L$
→ $Y_i = dy / L$
$X_{new} = X_1$; $Y_{new} = Y_1$.

Step 5: Initialize the point.
while (i<=L) {
$X_{new} += X_i$
$Y_{new} += Y_i$

Plot (Integer(X_{new}), Integer(Y_{new})) ; i++; }.

PROGRAM:

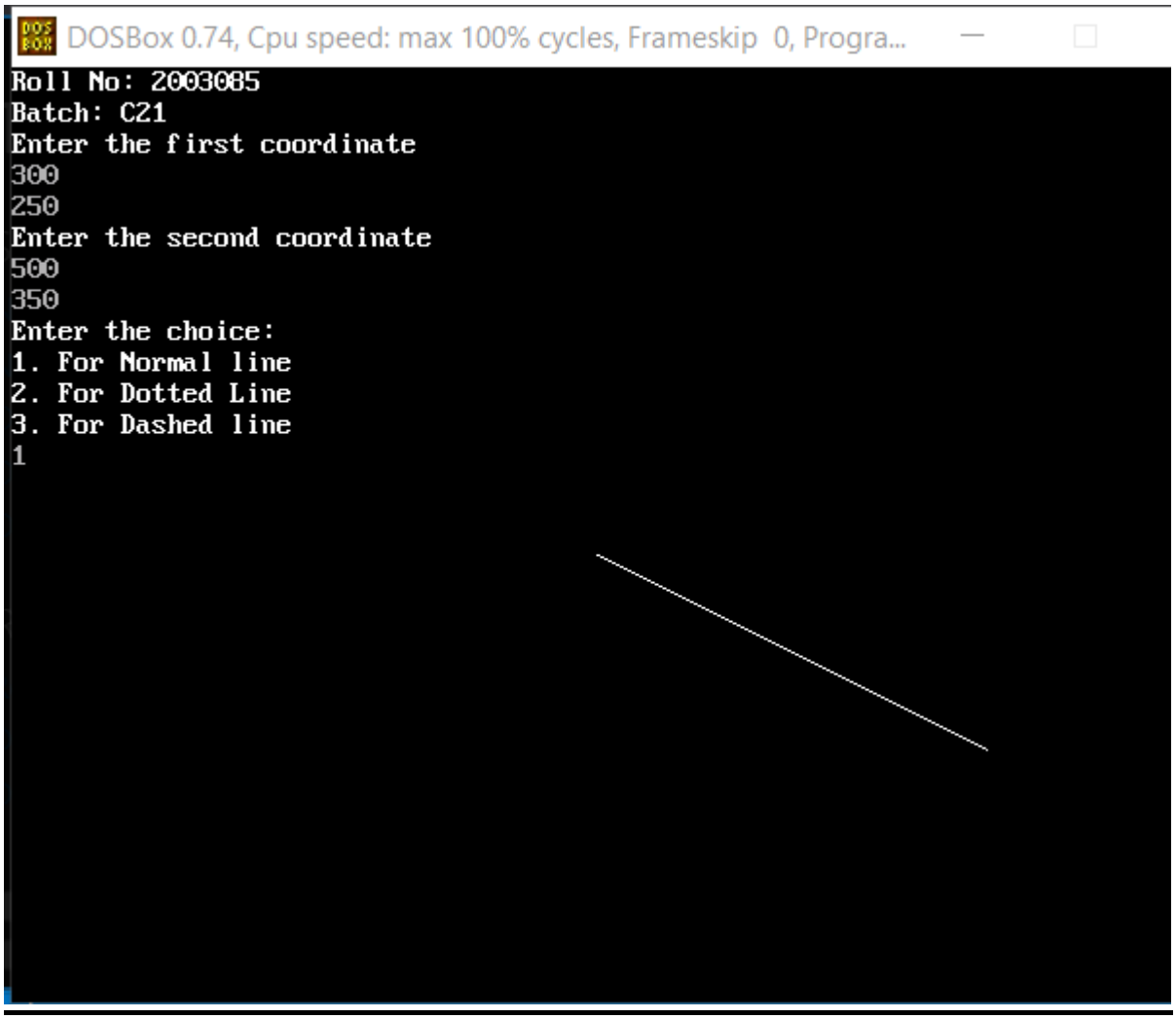
```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>

void main()
{
        int gd = DETECT, gm;
        float x, y, dx, dy, xi, yi, steps;
        int i, x1, y1, x2, y2, ch;
        initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
        printf("Roll No: 2003085 \nBatch: C21 \n");
        printf("Enter the first coordinate\n");
        scanf("%d %d", &x1, &y1);
        printf("Enter the second coordinate\n");
        scanf("%d %d", &x2, &y2);
        dx = abs(x2-x1);
        dy = abs(y2-y1);
        if(dx <= dy)
                steps = dy;
        else
                steps = dx;
        xi = dx/steps;
        yi = dy/steps;
        i = 0;
        x = x1; y = y1;
        printf("Enter the choice: \n");
        printf("1. For Normal line \n2. For Dotted Line \n3. For Dashed line \n");
```

```
scanf("%d", &ch);
switch(ch){
case 1:{
while(i <= steps){
        putpixel(x, y, 15);
        x += xi;
        y += yi;
        i++;}
break;
}
case 2:{
while(i <= steps){
        if(i%10==0)
        putpixel(x, y, 15);
        x += xi;
        y += yi;
        i++;}
break;
}
case 3:{
while(i <= steps){
        if(i%6!=4 && i%6!=5)
        putpixel(x, y, 15);
        x += xi;
        y += yi;
        i++;}
break;
}
default:
```

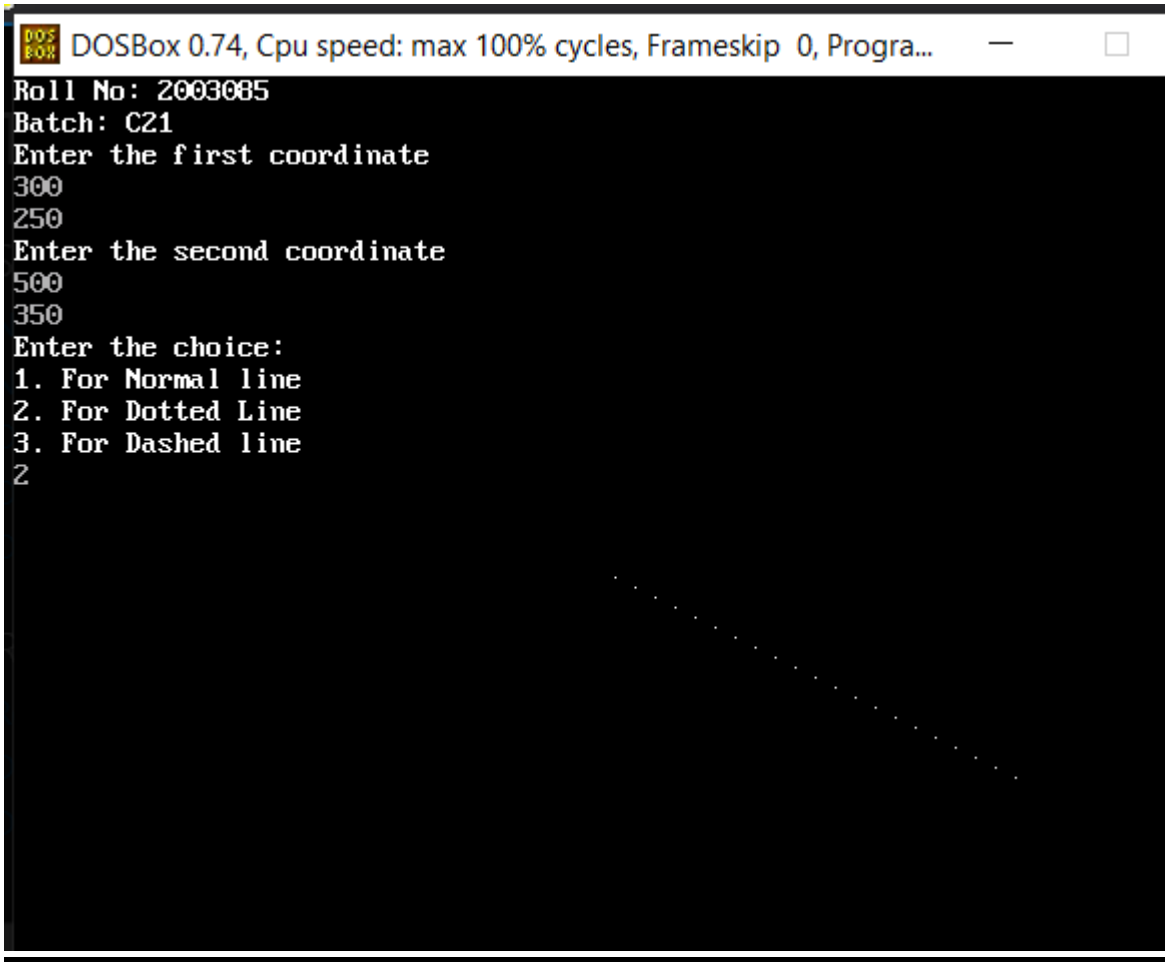
```
        printf("Invalid Choice!\n");


}
        getch();
}
```

OUTPUT:

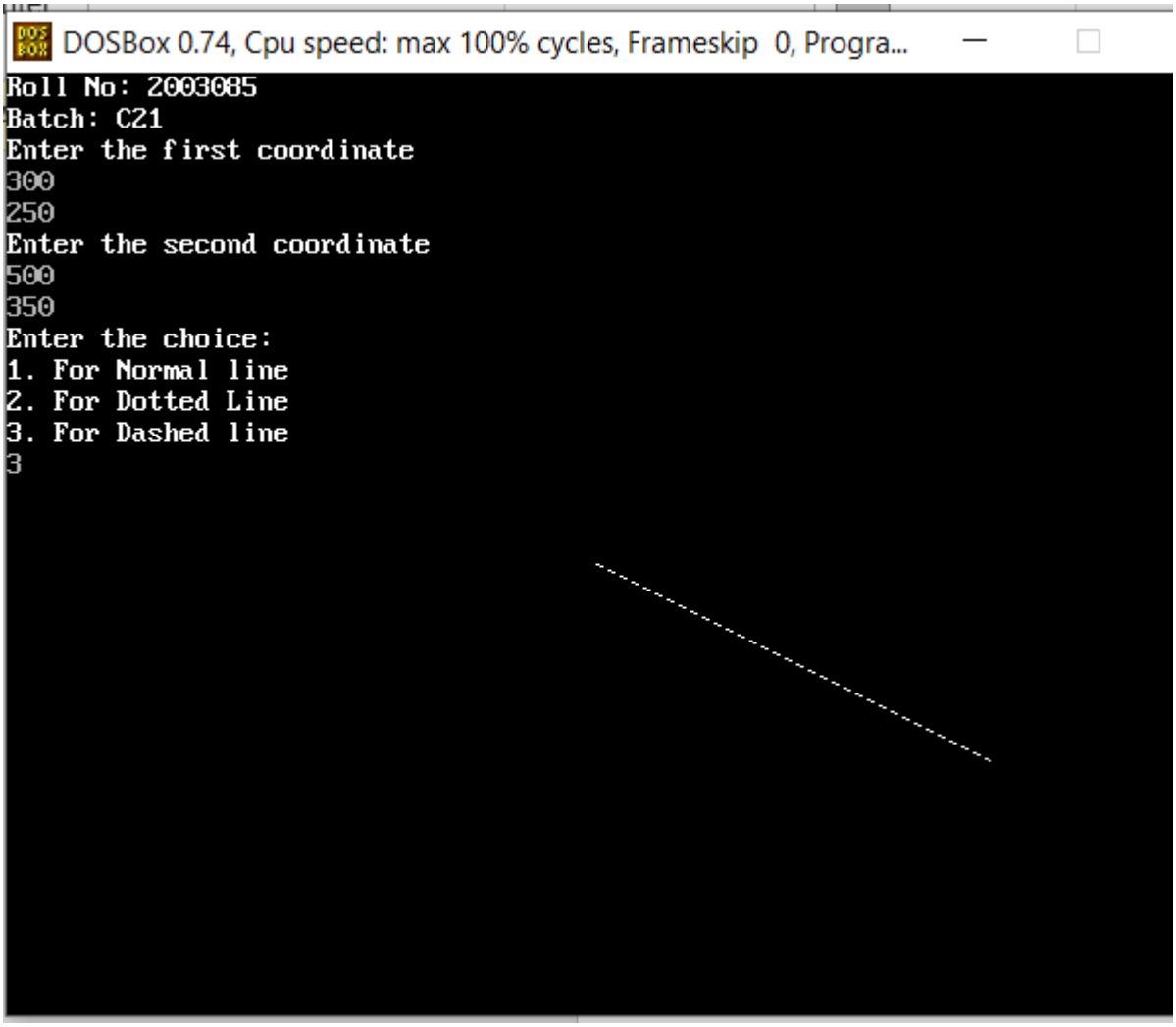
1. Normal Line:



2. Dotted Line:



3. Dashed Line



Harsh Kaushal
2003085
C21

Page No.
Date

Experiment No. 2

Aim:

Implement generalized Bresenham's line drawing algorithm in C.

Algorithm:

Step 1: Read (x_1, y_1) and (x_2, y_2)

Step 2: Calculate $dx = |x_2 - x_1|$ and $dy = |y_2 - y_1|$ where $|a|$ is absolute value of a.

Step 3: Initialize $x = x_1$ and $y = y_1$.

Step 4: Calculate $s_1 = sign(x_2 - x_1)$ where sign is a function and $s_2 = sign(y_2 - y_1)$. that returns -1, 0 & 1

Step 5: If $dy > dx$
then swap dy and dx and set exchange = 1
else set exchange = 0.

Step 6: Calculate initial decision parameter $e = 2dy - dx$.

Step 7: Set $i = 1$

Step 8: Plot (x, y)

Step 9: while $(e \geq 0)$
if (exchange = 1)
then $x = x + s_1$
else $y = y + s_2$
$e = e - 2dx$

Step 10: if (exchange = 1)
then $y = y + s_2$
else $x = x + s_1$
$e = e + 2dy$.

Step 11: $i = i + 1$

Step 12: if $(i \leq dx)$ goto step 8.

PROGRAM:

Code:

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
void Line(int, int, int, int);
void DashedLine(int, int, int, int);
void DottedLine(int, int, int, int);
void ThickLine(int, int, int, int);
int sign(int);
void main()
{
 int gd = DETECT, gm;
 int x1, y1, x2, y2, ch;
 initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
 printf("Enter the coordinates of the first point of the line\n");
 scanf("%d %d", &x1, &y1);
 printf("Enter the coordinates of the second point of the line\n");
 scanf("%d %d", &x2, &y2);
 printf("Enter the type of line you want to be drawn\n");
 printf("1-Solid\n2-Dotted\n3-Dashed\n4-Thick\n");
 scanf("%d", &ch);
 switch(ch)
 {
 case 1 :
 Line(x1, y1, x2, y2);
 break;
 case 2 :
```

```
 DottedLine(x1, y1, x2, y2);
 break;
 case 3:
 DashedLine(x1, y1, x2, y2);
 break;
 case 4:
 ThickLine(x1, y1, x2, y2);
 break;
 default:
 printf("INVALID INPUT!\n");
 }
 getch();
}
void Line(int x1, int y1, int x2, int y2)
{
 int dx, dy, s1, s2, exchange, temp, e, i, x, y;
 dx = abs(x2-x1);
 dy = abs(y2-y1);
 s1 = sign(x2-x1);
 s2 = sign(y2-y1);
 if(dy > dx)
 {
 exchange = 1;
 temp = dx;
 dx = dy;
 dy = temp;
 }
 else
 exchange = 0;
```

```
 e = 2*dy-dx;
 i = 1;
 x = x1;
 y = y1;
 while(i <= dx)
 {
 putpixel(x, y, 15);
 while(e >= 0)
 {
 if(exchange == 1)
 x += s1;
 else
 y += s2;
 e = e-2*dx;
 }
 if(exchange == 1)
 y += s2;
 else
 x += s1;
 e = e+2*dy;
 i++;
 }
}
void DottedLine(int x1, int y1, int x2, int y2)
{
 int dx, dy, s1, s2, exchange, temp, e, i, x, y;
 dx = abs(x2-x1);
 dy = abs(y2-y1);
 s1 = sign(x2-x1);
```

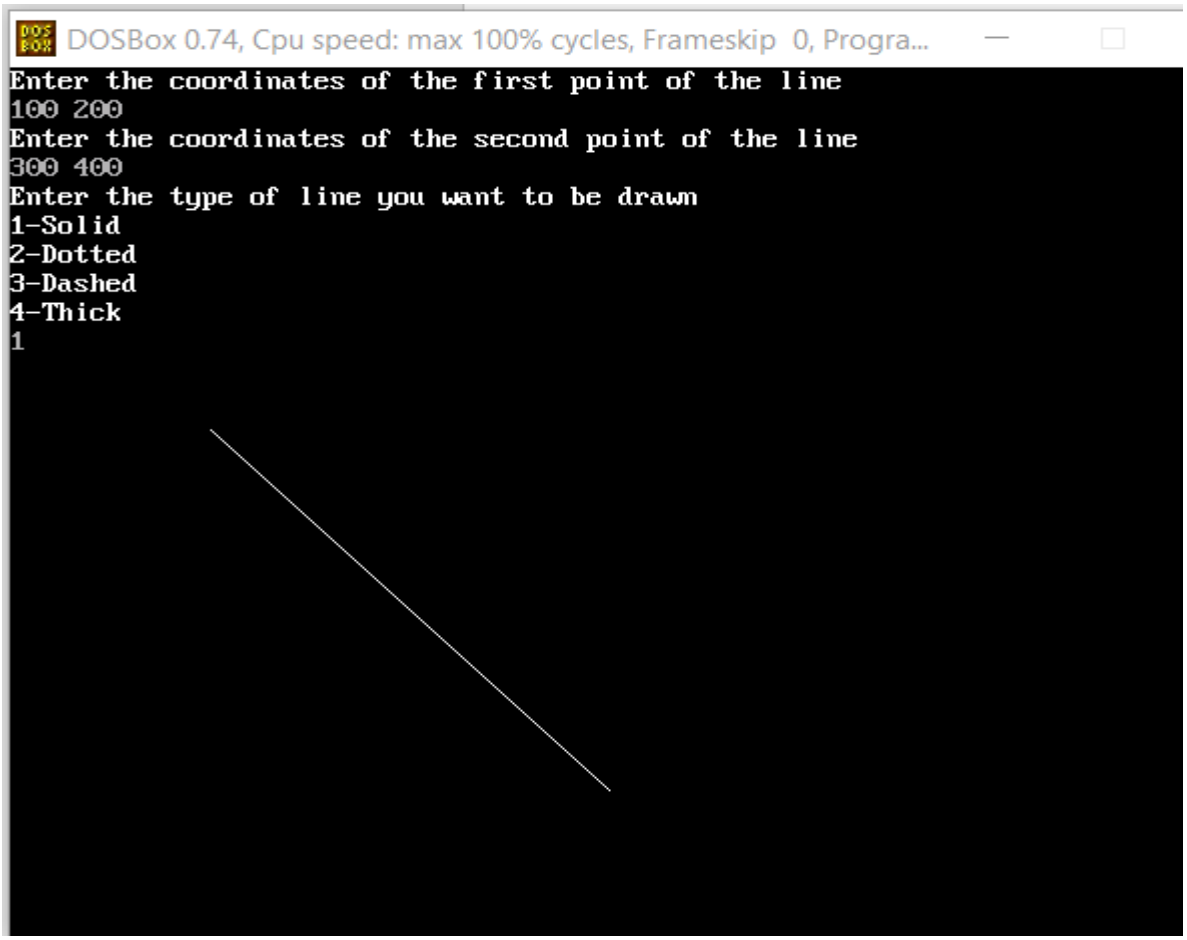
```
s2 = sign(y2-y1);
if(dy > dx)
{
exchange = 1;
temp = dx;
dx = dy;
dy = temp;
}
else
exchange = 0;
e = 2*dy-dx;
i = 1;
x = x1;
y = y1;
while(i <= dx)
{
if(i % 2 == 0)
putpixel(x, y, 15);
while(e >= 0)
{
if(exchange == 1)
x += s1;
else
y += s2;
e = e-2*dx;
}
if(exchange == 1)
y += s2;
else
```

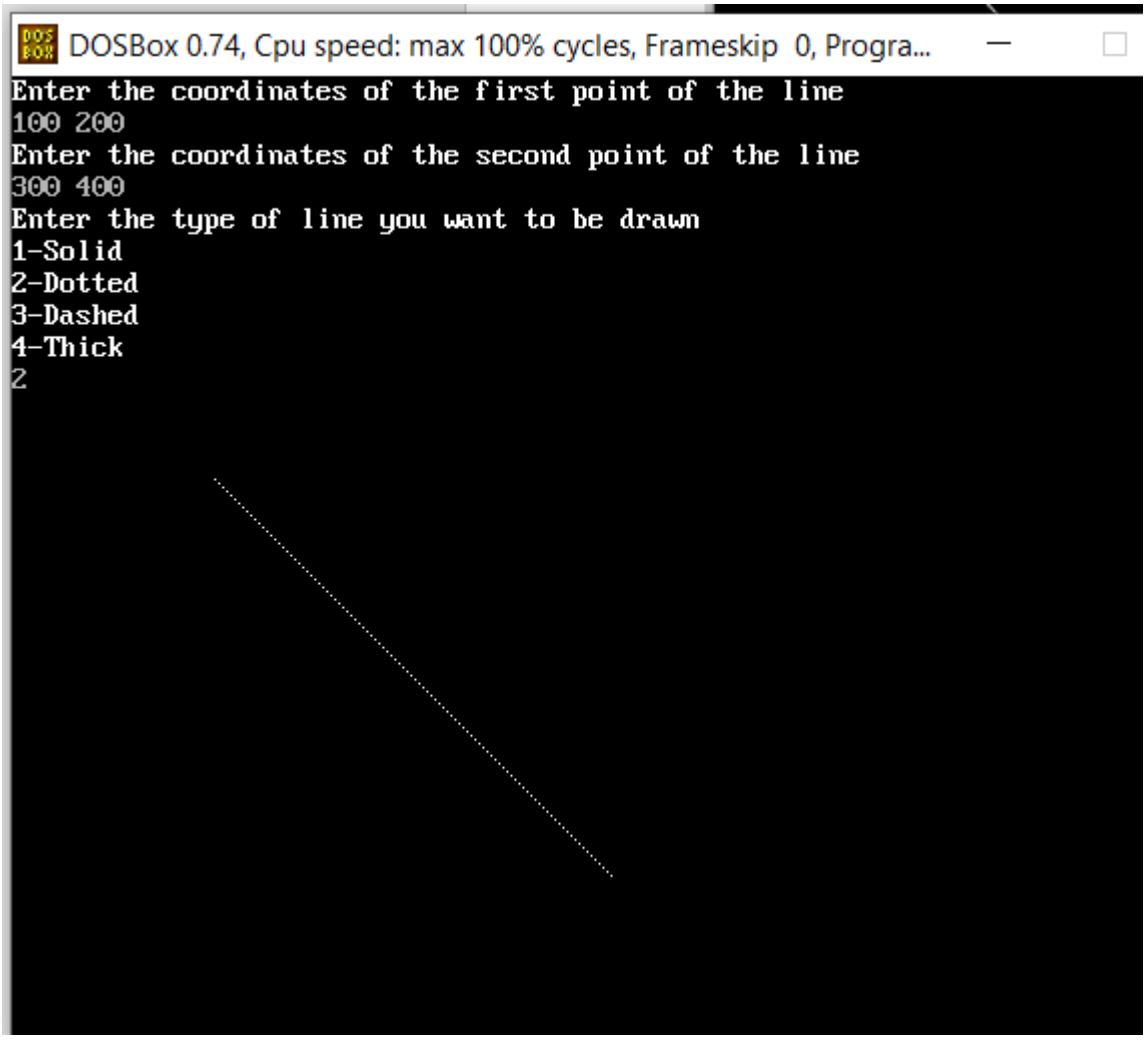
```
 x += s1;
 e = e+2*dy;
 i++;
 }
}
void DashedLine(int x1, int y1, int x2, int y2)
{
 int dx, dy, s1, s2, exchange, temp, e, i, x, y;
 dx = abs(x2-x1);
 dy = abs(y2-y1);
 s1 = sign(x2-x1);
 s2 = sign(y2-y1);
 if(dy > dx)
 {
 exchange = 1;
 temp = dx;
 dx = dy;
 dy = temp;
 }
 else
 exchange = 0;
 e = 2*dy-dx;
 i = 1;
 x = x1;
 y = y1;
 while(i <= dx)
 {
 if(i % 6 != 4 && i % 6 != 5)
 putpixel(x, y, 15);
```

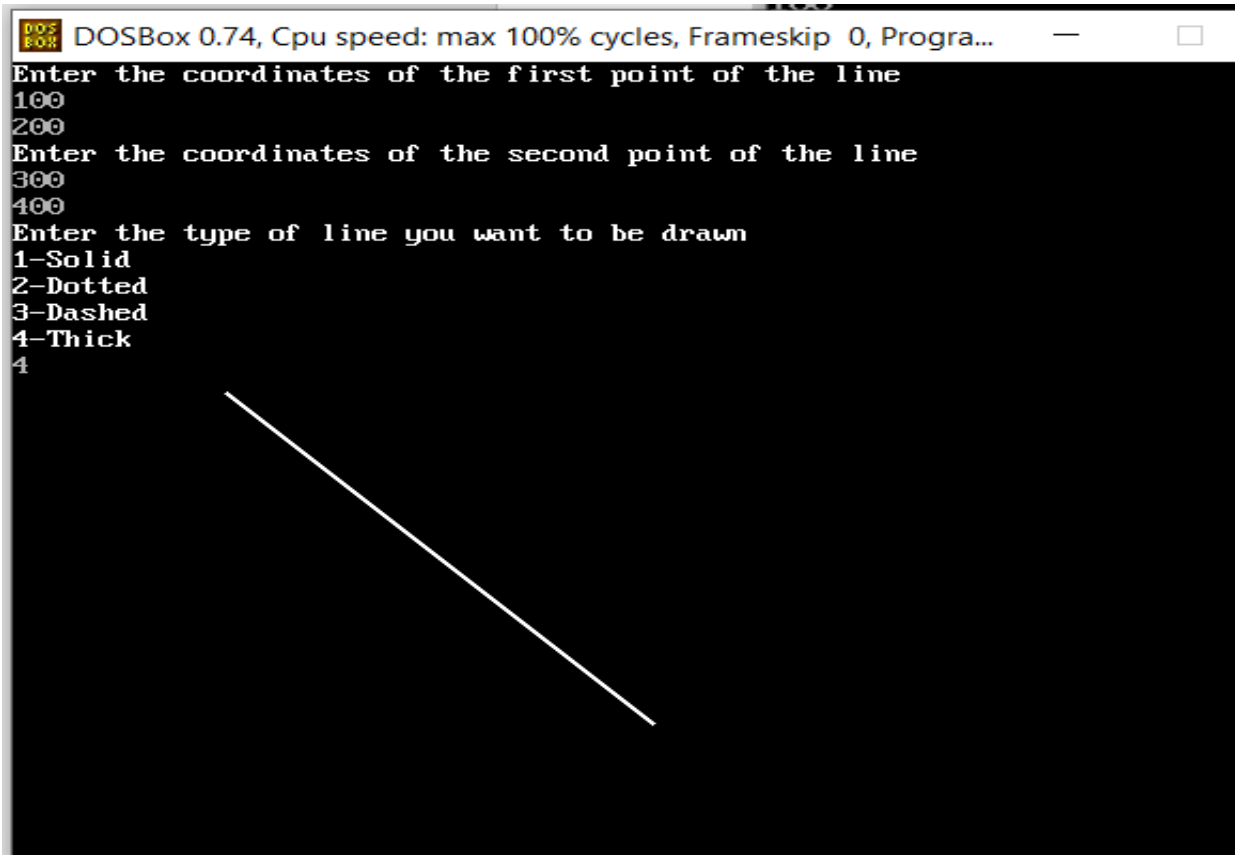
```
 while(e >= 0)
 {
 if(exchange == 1)
 x += s1;
 else
 y += s2;
 e = e-2*dx;
 }
 if(exchange == 1)
 y += s2;
 else
 x += s1;
 e = e+2*dy;
 i++;
 }
}
void ThickLine(int x1, int y1, int x2, int y2)
{
 Line(x1-1, y1, x2-1, y2);
 Line(x1, y1-1, x2, y2-1);
 Line(x1, y1, x2, y2);
 Line(x1+1, y1, x2+1, y2);
 Line(x1, y1+1, x2, y2+1);
}
int sign(int n)
{
 if(n > 0)
 return 1;
 else if(n < 0)
```

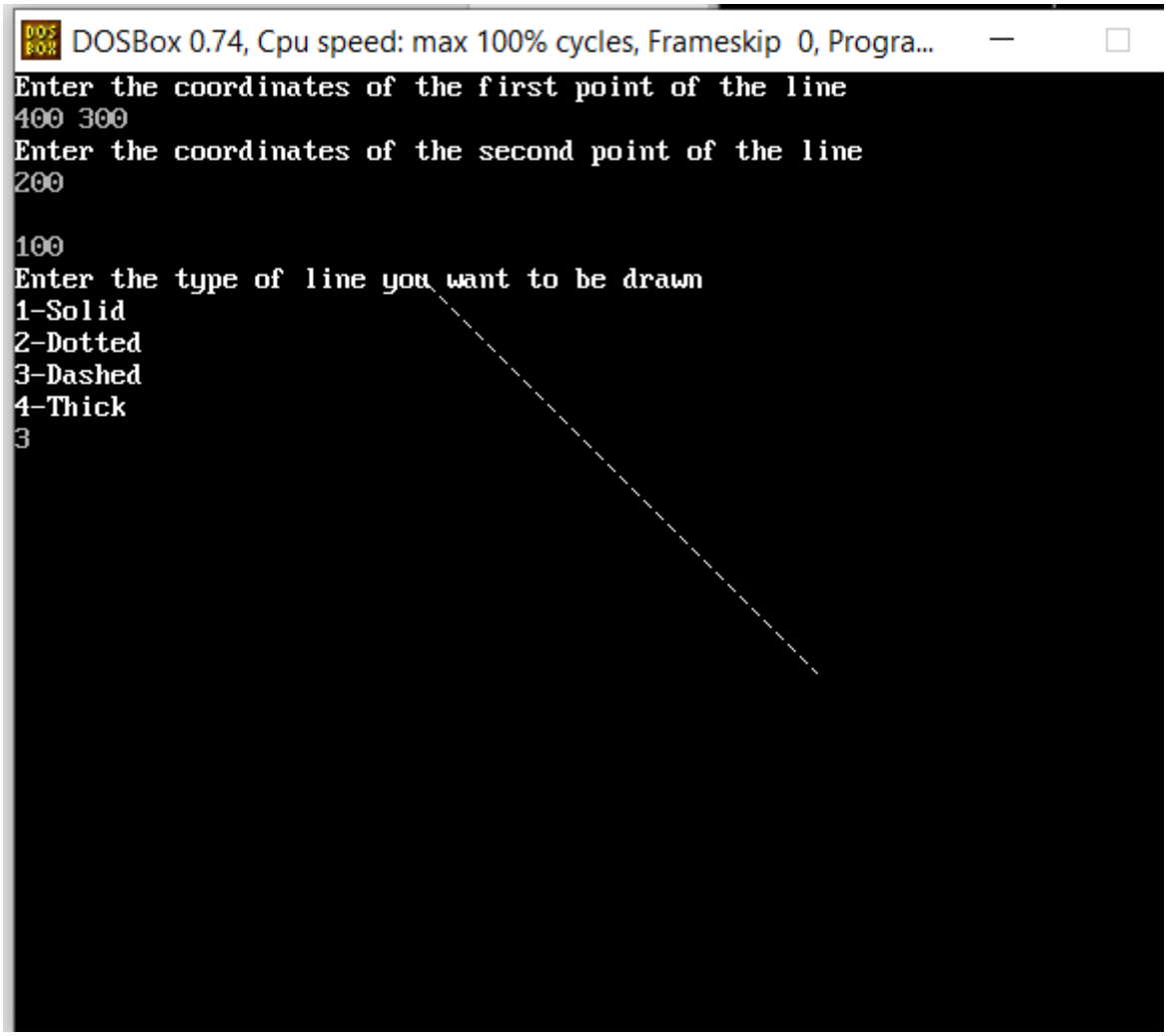
```
 return -1;
 return 0;
}
```

OUTPUT:




DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
Enter the coordinates of the first point of the line
100 200
Enter the coordinates of the second point of the line
300 400
Enter the type of line you want to be drawn
1-Solid
2-Dotted
3-Dashed
4-Thick
2


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
Enter the coordinates of the first point of the line
100
200
Enter the coordinates of the second point of the line
300
400
Enter the type of line you want to be drawn
1-Solid
2-Dotted
3-Dashed
4-Thick
4


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
Enter the coordinates of the first point of the line
400 300
Enter the coordinates of the second point of the line
200
100
Enter the type of line you want to be drawn
1-Solid
2-Dotted
3-Dashed
4-Thick
3

Harsh Kashiwal
2003085
C21

Page No.
Date

Experiment No. 13

Aim: Implement midpoint circle and midpoint ellipse drawing algorithm in C.

Theory:

Circle: Midpoint circle drawing algorithm works by finding out whether the midpoint of two pixels between which the circle passes through is above or below the circumference of the circle. Depending on that the selection of pixel is made. Also, eight-octant symmetry is used to draw the whole circle and only one octant's pixels are actually calculated.

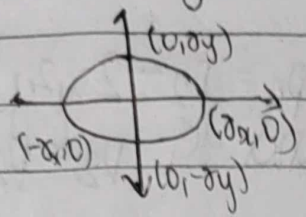
Algorithm:

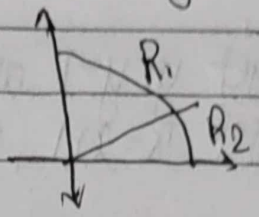
step 1: Take radius (r) and centre of circle (x_c, y_c) as input.
step 2: set $x=0, y=r$.
step 3: calculate initial decision parameter $p=1-r$.
step 4: Plot the 8 pixels in the octants that is,

(x_c+x, y_c+y), (x_c+x, y_c-y)
(x_c-x, y_c+y), (x_c-x, y_c-y)
(x_c+y, y_c+x), (x_c+y, y_c-x)
(x_c-y, y_c+x), (x_c-y, y_c-x).

step 5: If $p<0$
then set $x=x+1$
and set $p=p+2x+1$
else set $x=x+1$ and $y=y-1$
and set $p=p+2x+1-2y$.
step 6: Repeat steps 4-5 until $x \geq y$.

Ellipse: Ellipse is an elongated circle which can be described with a centre and two radii. The radius of the ellipse along x-axis (r_x) and radius of ellipse along y-axis (r_y). In midpoint ellipse drawing, we need to make use of 4 quadrant symmetry instead of 8-octant symmetry. We will call the region of ellipse with slope < -1 region 1 and slope > -1, region 2





Algorithm:

Step 1: Take centre of ellipse (x_c, y_c) and radii (r_x and r_y)

Step 2: Set $x = 0$, $y = r_y$

Step 3: Calculate initial decision parameter in region 1.

$$p_1 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

Step 4: Plot the 4 pixels in each quadrant, that is

$(x_c - x, y_c + y)$, $(x_c + x, y_c + y)$.

$(x_c - x, y_c - y)$, $(x_c + x, y_c - y)$.

Step 5: If $p_1 < 0$

then set $x = x + 1$ and set $p_1 = p_1 + 2r_y^2 x + r_y^2$

else set $x = x + 1$; $y = y - 1$ and set $p_1 = p_1 + 2r_y^2 x - 2r_x^2 y + r_y^2$

Step 6: Repeat steps 4-5 until $2r_y^2 x \geq 2r_x^2 y$.

Now the first region has been plotted.

Step 7: Calculate initial decision parameter in region 2 using x and y values reached in region 1.

$$p_2 = r_y^2\left(\frac{1}{2}+x\right)^2 + r_x^2(y-1)^2 - r_x^2 r_y^2$$

Step 8: Plot the 4 points in each quadrant, same as region 1.

Step 9: If $p_2 > 0$,

then set $y = y-1$ and set $p_2 = p_2 - 2r_y^2 y + r_x^2$

else set $x = x+1$, $y = y-1$ and set $p_2 = p_2 + 2r_y^2 x - 2r_x^2 y + r_x^2$

Step 10: Repeat steps 8-9 until $2r_y^2 x \geq 2r_x^2 y$

Now both regions 1 and 2 have been plotted and thus complete ellipse has been plotted.

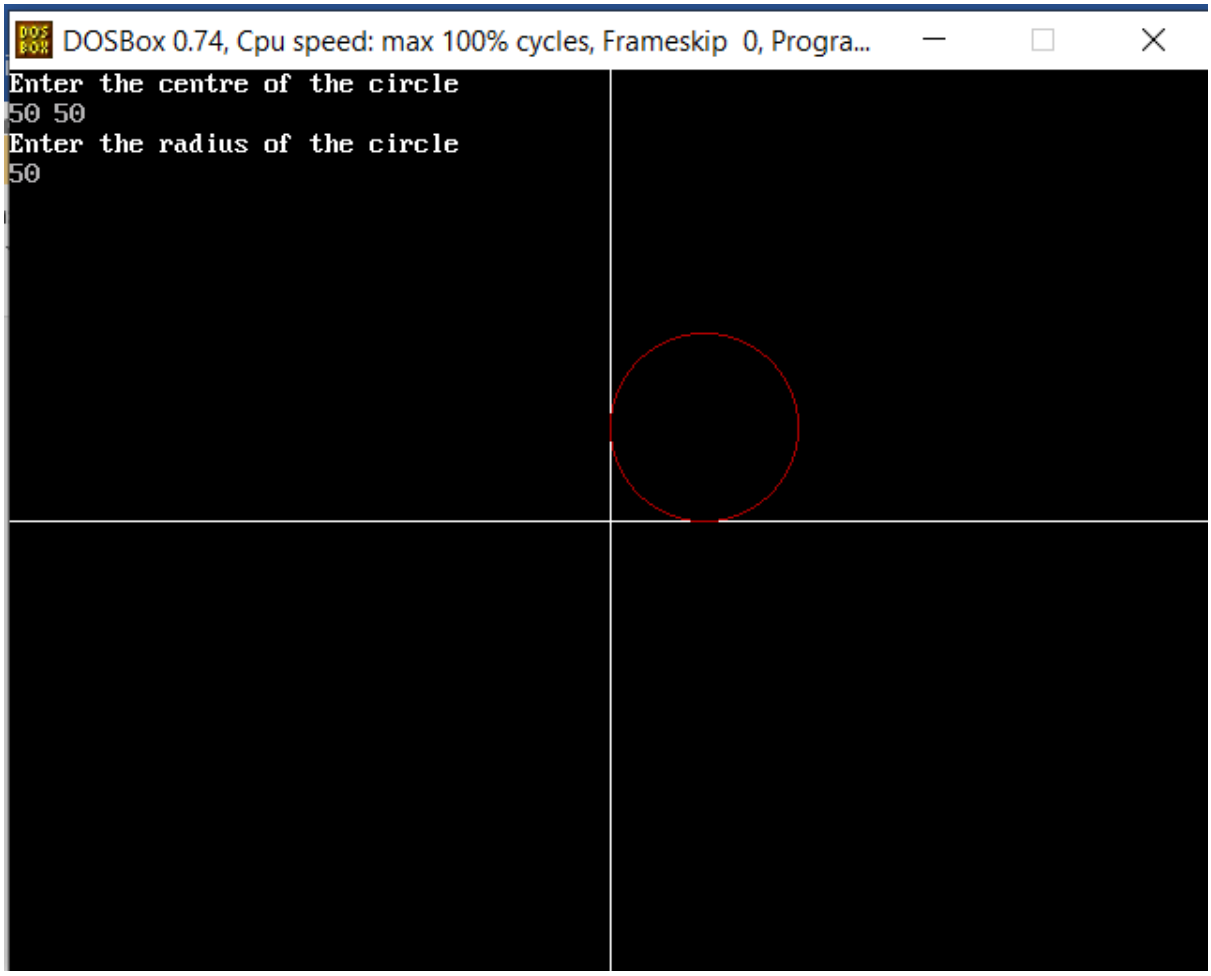
PROGRAM:

Code 1: Midpoint Circle

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
void plot(int, int, int, int, int);
void Circle(int, int, int, int);
void main()
{
 int gd = DETECT, gm;
 int xc, yc, r, col = RED;
 initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
 line(320, 0, 320, 480);
 line(0, 240, 640, 240);
 printf("Enter the centre of the circle\n");
 scanf("%d %d", &xc, &yc);
 printf("Enter the radius of the circle\n");
 scanf("%d", &r);
 Circle(xc, yc, r, col);
 getch();
}
void Circle(int xc, int yc, int r, int col)
{
 int x = 0, y = r, p = 1-r;
 while(x <= y)
 {
 plot(xc, yc, x, y, col);
 if(p < 0)
```

```
 {
 x++;
 p = p + 2*x + 1;
 }
 else
 {
 x++;
 y--;
 p = p + 2*(x-y) + 1;
 }
 }
}
void plot(int xc, int yc, int x, int y, int col)
{
 putpixel(320+xc+x, 240-(yc+y), col);
 putpixel(320+xc+x, 240-(yc-y), col);
 putpixel(320+xc-x, 240-(yc+y), col);
 putpixel(320+xc-x, 240-(yc-y), col);
 putpixel(320+xc+y, 240-(yc+x), col);
 putpixel(320+xc+y, 240-(yc-x), col);
 putpixel(320+xc-y, 240-(yc+x), col);
 putpixel(320+xc-y, 240-(yc-x), col);
}
```

OUTPUT:

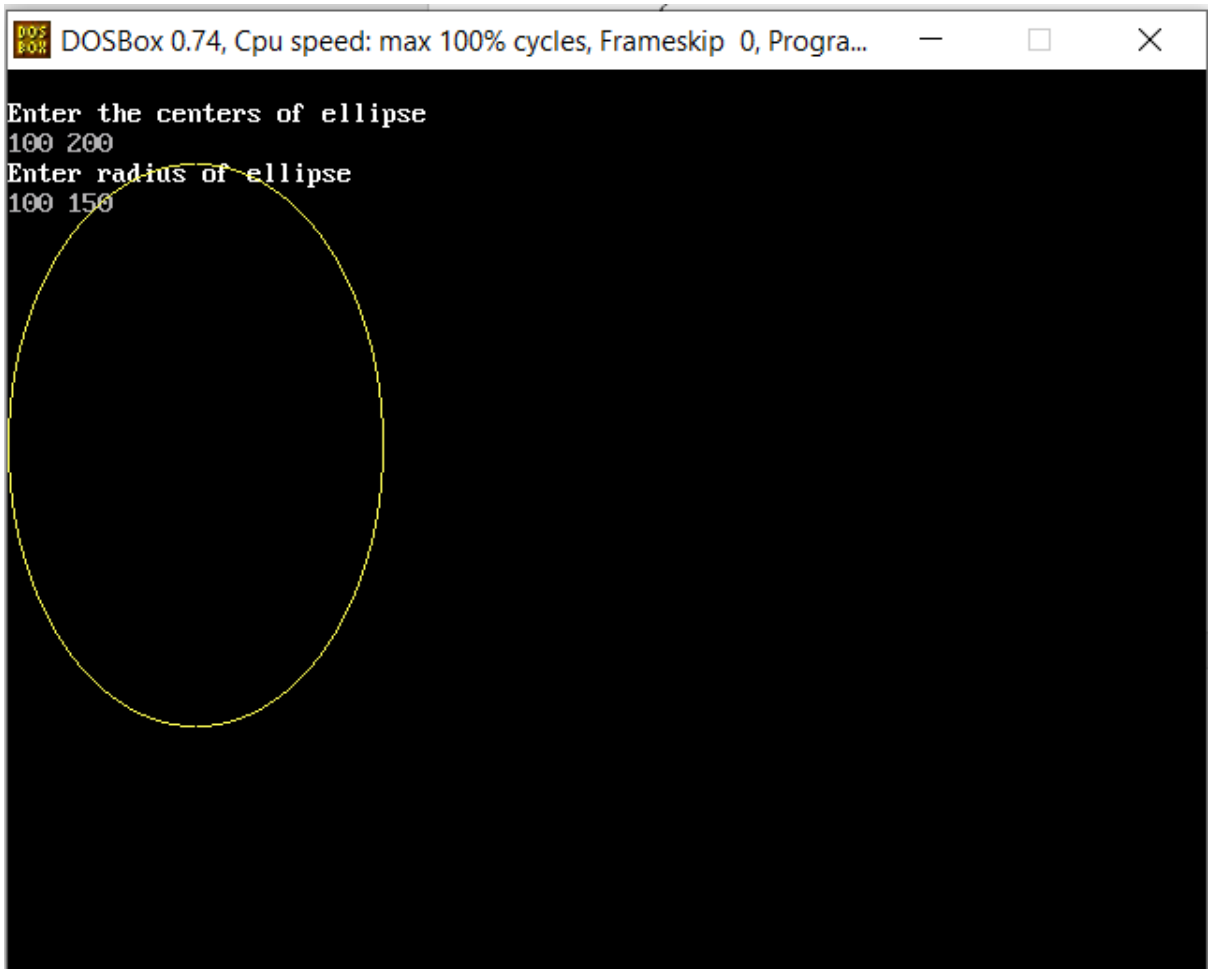

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
Enter the centre of the circle
50 50
Enter the radius of the circle
50

Code 2: Midpoint Ellipse

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<math.h>
void main(){
int gd=DETECT, gm;
float xc,yc,rx,ry,x,y,pk,p1k,p2k;
initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
printf("\nEnter the centers of ellipse\n");
scanf("%f%f", &xc, &yc);
printf("Enter radius of ellipse\n");
scanf("%f%f", &rx, &ry);
x=0; y=ry;
pk= (ry*ry)-(rx*rx*ry)+((rx*rx)/4);
while((2*ry*ry*x)<(2*rx*rx*y)){
        if(pk<=0){
        x++;
        p1k=pk+(2*ry*ry*x)+(ry*ry);
        }
        else{
        x++;
        y--;
        p1k=pk+(2*ry*ry*x)-(2*rx*rx*y)+(ry*ry);
        }
pk=p1k;
putpixel(xc+x, yc+y,14);
```

```
putpixel(xc-x, yc+y,14);
putpixel(xc+x, yc-y,14);
putpixel(xc-x, yc-y,14);
}
pk=((x+0.5)*(x+0.5)*ry*ry)+((y-1)*(y-1)*rx*rx)-(rx*rx*ry*ry);
while(y>0){
        if(pk>0){
        y=y-1;
        p2k=pk-(2*rx*rx*y)+(rx*rx);
        }
        else{
        x=x+1;
        y=y-1;
        p2k=pk+(2*ry*ry*x)-(2*rx*rx*y)+(rx*rx);
        }
pk=p2k;
putpixel(xc+x, yc+y,14);
putpixel(xc-x, yc+y,14);
putpixel(xc+x, yc-y,14);
putpixel(xc-x, yc-y,14);
}
getch();
}
```

OUTPUT:


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
Enter the centers of ellipse
100 200
Enter radius of ellipse
100 150

Harsh Kayliwal
2009035
C2)

Page No.
Date

Experiment 4

Aim: Implement midpoint drawing ellipse method in C.

Theory:

Algorithm:

1. Start.
2. Declare $r_x, r_y, x, y, m, dx, dy, p_1, p_2$.
3. Initialise point of region 1 as $x = 0$
   $y = r_y$.
4. Calculate
   $p = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$
   $dx = 2 r_y^2 x$.
   $dy = 2 r_x^2 y$.
5. Update values of dx, and dy after each iteration
6. Repeat steps while $(dx < dy)$
   plot (x, y)
   if $(p < 0)$
   update $x = y + 1$
   $p_1 = r_y^2 [2x + 3]$
   else
   update $x = x + 1$
   $y = y - 1$
7. when $dx > dy$, plot region 2
8. calculate $p_2 = r_y^2 (x + 1/2)^2 + r_x^2 (y - 1)^2 - r_x^2 r_y^2$

9. $xy > 0$
if ($P_2 > 0$)
update $y = y - 1$
$P_2 = P_2 - 2ry^2 + r$
else
$x = x + 1$
$y = y - 1$
$P_2 = P_2 + 2rx^2 [2x] - 2ry + r$

10. END

PROGRAM:

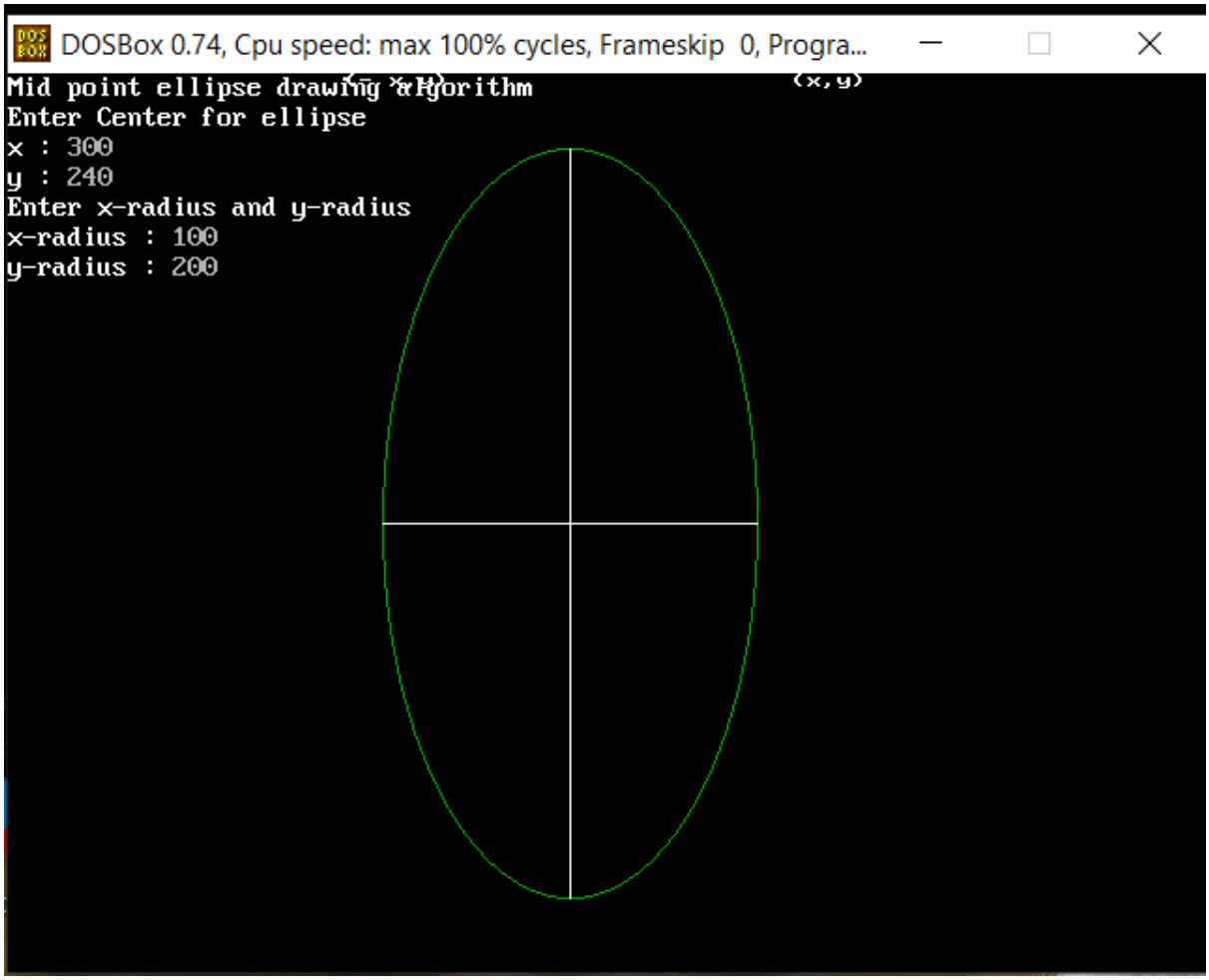
Code:

```
#include <conio.h>
#include <stdio.h>
#include <graphics.h>
void main()
{
int gd=DETECT,gm;
float x,y,xc,yc,rx,ry,pk,pk1;
clrscr();
initgraph(&gd,&gm,"..\\bgi");
printf("Mid point ellipse drawing algorithm\n");
printf("Enter Center for ellipse\nx : ");
scanf("%f",&xc);
printf("y : ");
scanf("%f",&yc);
printf("Enter x-radius and y-radius\nx-radius : ");
scanf("%f",&rx);
printf("y-radius : ");
scanf("%f",&ry);
x=0;
y=ry;
pk=(ry*ry)-(rx*rx*ry)+((rx*rx)/4);
while((2*x*ry*ry)<(2*y*rx*rx))
{
if(pk<=0)
{
```

```
x=x+1;
pk1=pk+(2*ry*ry*x)+(ry*ry);
}
else
{
x=x+1;
y=y-1;
pk1=pk+(2*ry*ry*x)-(2*rx*rx*y)+(ry*ry);
}
pk=pk1;
putpixel(xc+x,yc+y,2);
putpixel(xc-x,yc+y,2);
putpixel(xc+x,yc-y,2);
putpixel(xc-x,yc-y,2);
}
pk=((x+0.5)*(x+0.5)*ry*ry)+((y-1)*(y-1)*rx*rx)-(rx*rx*ry*ry);
while(y>0)
{
if(pk>0)
{
y=y-1;
pk1=pk-(2*rx*rx*y)+(rx*rx);
}
else
{
x=x+1;
y=y-1;
```

```
pk1=pk+(2*ry*ry*x)-(2*rx*rx*y)+(rx*rx);
}
pk=pk1;
putpixel(xc+x,yc+y,2);
putpixel(xc-x,yc+y,2);
putpixel(xc+x,yc-y,2);
putpixel(xc-x,yc-y,2);
}
line(xc+rx,yc,xc-rx,yc);
line(xc,yc+ry,xc,yc-ry);
outtextxy(xc+(1.2*rx),yc-(1.2*ry),"(x,y)");
outtextxy(xc-(1.2*rx),yc+(1.2*ry),"(- x,-y)");
outtextxy(xc+(1.2*rx),yc+(1.2*ry),"(x,- y)");
outtextxy(xc-(1.2*rx),yc-(1.2*ry),"(- x,y)");
getch();
}
```

OUTPUT:


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
Mid point ellipse drawing algorithm
Enter Center for ellipse
x : 300
y : 240
Enter x-radius and y-radius
x-radius : 100
y-radius : 200
(x,y)

Harsh Kushwaha
2003085
C2-1

Page No.
Date

Experiment 5

Aim: Implement flood fill and boundary fill to fill a polygon

Theory:

Flood fill is an algorithm mainly used to determine a bounded area connected to a given node in a multidimensional array. Boundary fill is the algorithm used frequently in computer graphics to fill a desired color inside a closed polygon

Algorithm:

Flood fill:

```
flood fill (x, y, old, color, new color){
 put pixel (x, y, new color)
 flood fill (x+1, y, old-color, new color)
 flood fill (x-1, y, old-color, new-color)
 flood fill (x, y+1, old, new-color)
 flood fill (x, y-1, old color, new-color);
 flood fill (x+1, y+1, old color, new color);
 flood fill (x-1, y-1, old color, new color);
 flood fill (x-1, y-1, old-c, new-c);
 flood fill (x-1, y+1, old-c, new c);
}
```

Boundary fill algorithm:

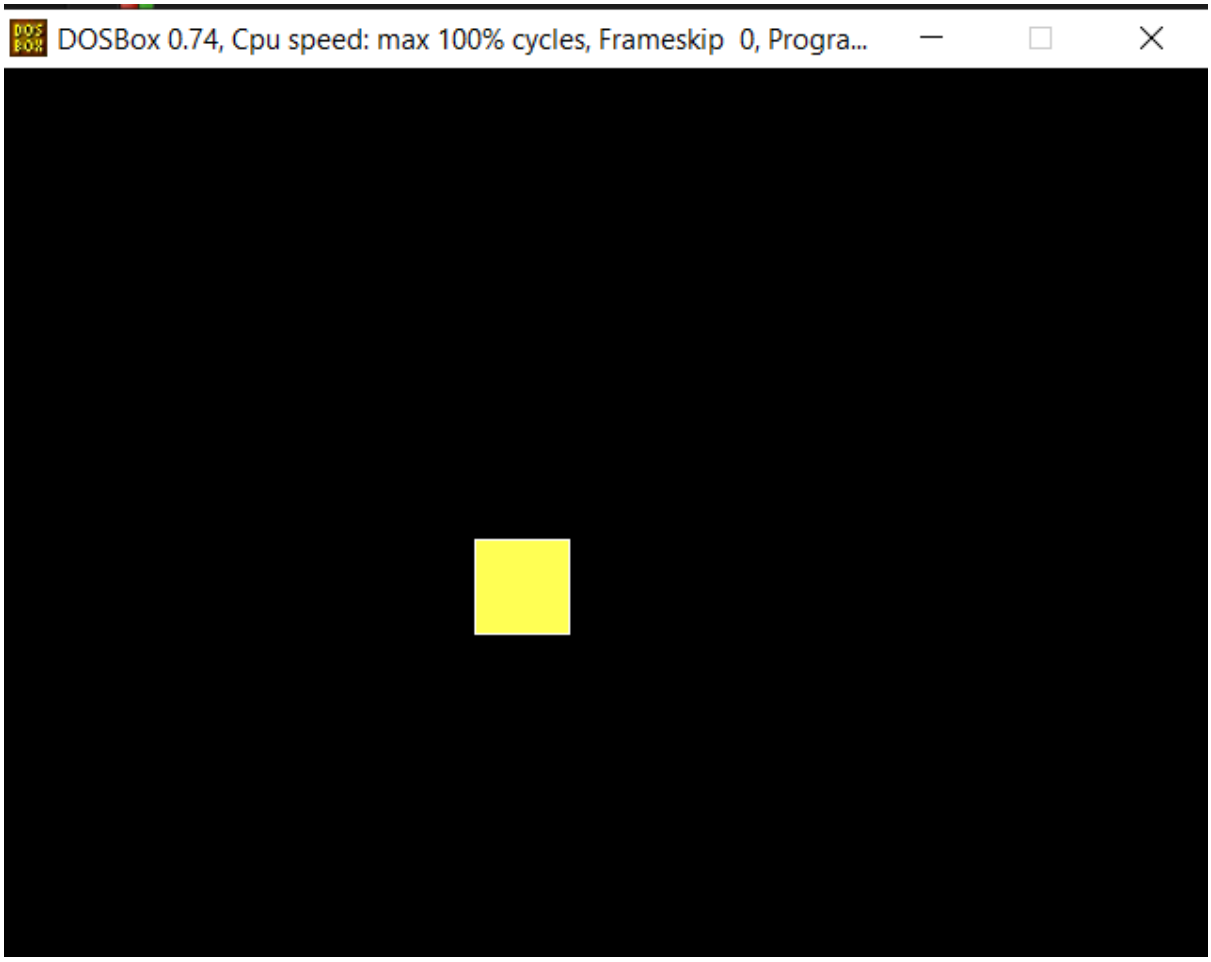
```
boundaryfill(x, y, f-color, b-color){
    if(getpixel(x,y) != b-color && getpixel(x,y) != f-color)
        putpixel(x,y, f-color)
    boundaryfill(x+1, y, f-color, b-color);
    boundaryfill(x, y+1, f-color, b-color);
    boundaryfill(x-1, y, f-color, b-color);
    boundaryfill(x, y-1, f-color, b-color);
}
```

PROGRAM:

Code: Flood Fill

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>

void fooldfill(int x, int y, int fill, int old){
	if(getpixel(x,y)==old){
		putpixel(x,y,fill);
		fooldfill(x+1,y,fill,old);
		fooldfill(x,y+1,fill,old);
		fooldfill(x-1,y,fill,old);
		fooldfill(x,y-1,fill,old);
	}
}
int main(){
	int gm, gd=DETECT;
	int x,y;
	initgraph(&gd,&gm,"c:\\turboc3\\bgi");
	rectangle(250,250,300,300);
	fooldfill(260,260,YELLOW,0);
	delay(5000);
	getch();
	closegraph();
	return 0;
}
```

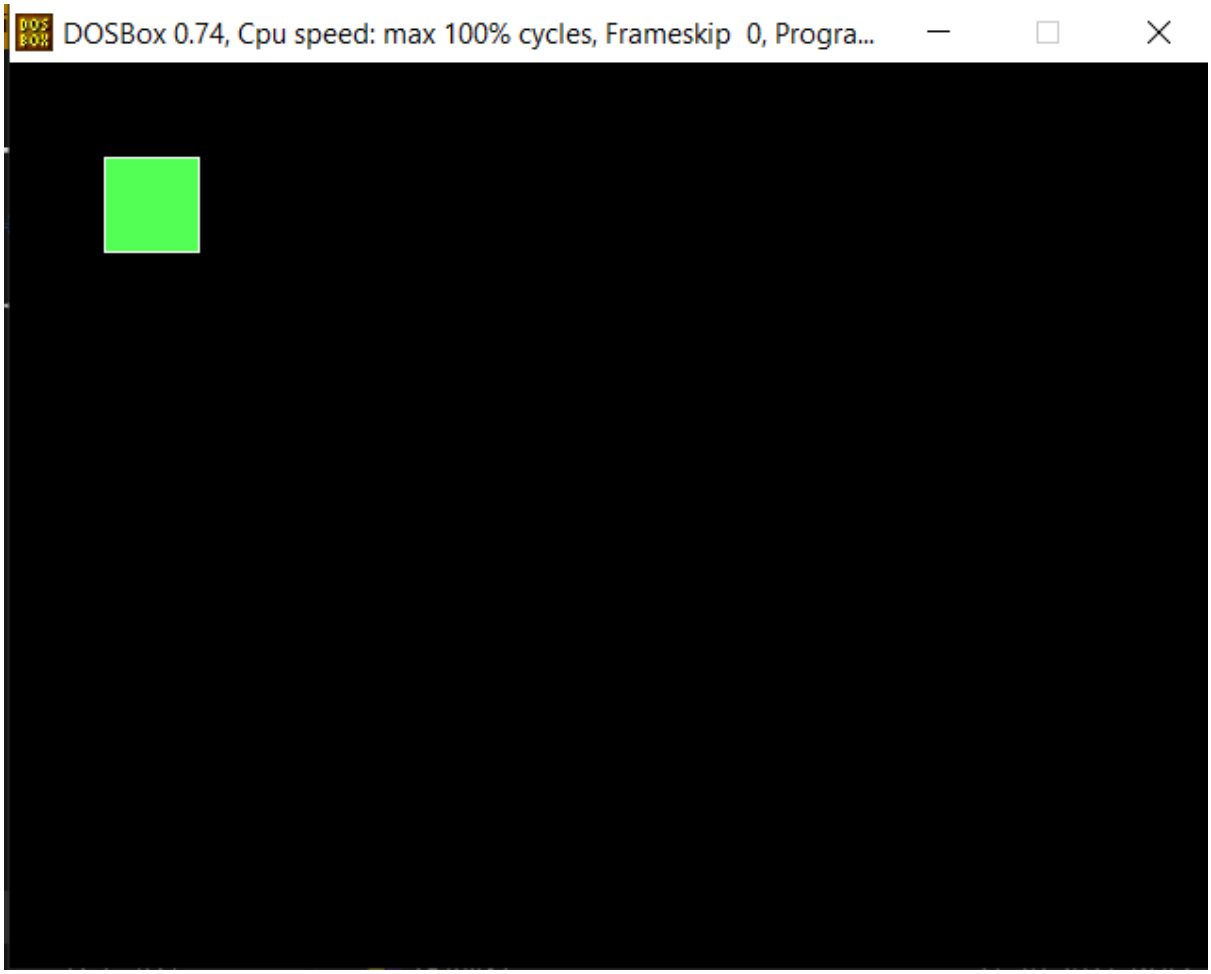
OUTPUT:



Code 2: Boundary Fill

```
#include<stdio.h>
#include<graphic.h>
void boundaryfill(int x, int y, int fill, int boundary){
        if(getpixel(x,y)!=boundary && getpixel(x,y)!=fill){
        putpixel(x,y,fill);
        boundaryfill(x+1,y,fill,boundary);
        boundaryfill(x-1,y,fill,boundary);
        boundaryfill(x,y+1,fill,boundary);
        boundaryfill(x,y-1,fill,boundary);
        boundaryfill(x+1,y+1,fill,boundary);
        boundaryfill(x+1,y-1,fill,boundary);
        boundaryfill(x-1,y+1,fill,boundary);
        boundaryfill(x-1,y-1,fill,boundary);
        }
}

int main(){
        int gm, gd=DETECT;
        int x,y;
        initgraph(&gd,&gm,"c:\\turboc3\\bgi");
        rectangle(100,100,50,50);
        boundaryfill(55,55,10,15);
        delay(5000);
        getch();
        closegraph();
        return 0;
}
```

OUTPUT:



Harsh Kashiwal
2003055
(2)

Page No.
Date

Experiment 6

Aim: Implement 2D transformations on a polygon translation, Rotation, Scaling, Reflection, Shearing.

Theory:-

1) Translation: To translate a point from co-ordinate position (x, y) to another (x_1, y_1) we add algebraically the translation distance T_x and T_y to original co-ordinate.

$$x_1 = x + T_x$$
$$y_1 = y + T_y$$

Matrix for Translation:
$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

2) Scaling: It is used to alter or change the size of objects. The change is done using scaling factors. There are two scaling factors i.e S_x in x direction and S_y in y-direction.

Matrix for scaling:
$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3) Rotation: Process of changing the angle of the object. Rotation can be clockwise or anticlockwise. For rotation we have to specify the angle of rotation and point.

Rotation about an arbitrary point: If we want to rotate an object or point about an arbitrary point, first of all, we translate the point about which we want to rotate to the origin. Then rotate point or object about the origin, and at the end, we again translate it to the original place. We get rotation about an arbitrary point.

Matrix for homogenous co-ordinate rotation:

Clockwise

$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Anticlockwise.

$$R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4> Reflection: It is a transformation which produces a mirror image of an object. The mirror image can be either about x or y axis.

=> Reflection about x-axis:

In this transformation value of x will remain same where the value of y will become negative. Object can be reflected about x axis with following matrix.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

⇒ Reflection about Y axis: Here the values of x will be reversed the value of y will remain same. Object can be reflected about Y axis with following matrix: $\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

⇒ Shearing: It is a transformation which changes the shape of the object. The sliding of layers of objects occur. The shear can be in one direction or in two directions.

⇒ Shearing in the x direction
In this horizontal shearing sliding of layer occurs. The homogenous matrix for shearing in the x-direction is shown below.

$$\begin{bmatrix} 1 & 0 & 0 \\ sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

⇒ Shearing in the y direction: Here shearing is done by sliding along vertical or Y axis.

$$\begin{bmatrix} 1 & sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

⇒ Shearing in x-y direction: Here layers will be slided in both x as well as y direction. The sliding will be in horizontal as well as vertical direction. The shape of the object will be distorted. The matrix of shear in both directions is given by.

$$\begin{bmatrix} 1 & sh_x & 0 \\ sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

PROGRAM:

Code:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <graphics.h>

void multiply(float[3][3], float[3]), clearMat(float[3][3]);
void translate(float[3], float, float);
void rotate(float[3], float);
void scale(float[3], float, float);
void reflectX(float[3]), reflectY(float[3]), reflect(float[3]);
void shear(float[3], float, float);
void Triangle(float[3], float[3], float[3]);
        void main() {
        float p1[3] = {0, 0, 1}, p2[3] = {0, 0, 1}, p3[3] = {0, 0, 1};
        float tx, ty, sx, sy, shx, shy, theta;
        int ch;
        int gd = DETECT, gm;
        initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
        line(320, 0, 320, 480); //y-axis
        line(0, 240, 640, 240); //x-axis
        printf("Enter cordinates of first point of triangle\n");
        scanf("%f %f", &p1[0], &p1[1]);
        printf("Enter coordinates of second point of triangle\n");
        scanf("%f %f", &p2[0], &p2[1]);
        printf("Enter coordinates of third point of triangle\n");
        scanf("%f %f", &p3[0], &p3[1]);
```

```
        Triangle(p1, p2, p3);
        printf("Enter your choice\n1-Translate\n2-Rotate\n3-Scale\n4-Reflect\n5-
Shear\n");
        scanf("%d", &ch);
 switch(ch){
 case 1:
        printf("Enter value of translation value in x direction\n");
        scanf("%f", &tx);
        printf("Enter translation value in y direction\n");
        scanf("%f", &ty);
        translate(p1, tx, ty);
        translate(p2, tx, ty);
        translate(p3, tx, ty);
        Triangle(p1, p2, p3);
        break;
 case 2:
        printf("Enter angle of rotation in degrees\n");
        scanf("%f", &theta);
        theta *= 3.1415f/180;
        rotate(p1, theta);
        rotate(p2, theta);
        rotate(p3, theta);
        Triangle(p1, p2, p3);
        break;
 case 3:
        printf("Enter scale value in x direction\n");
        scanf("%f", &sx);
        printf("Enter scale value in y direction\n");
        scanf("%f", &sy);
```

```
        scale(p1, sx, sy);
        scale(p2, sx, sy);
        scale(p3, sx, sy);
        Triangle(p1, p2, p3);
        break;
case 4:
        printf("Reflection about what ?\n");
        printf("1 - X axis\n2- Y axis\n3 - Origin");
        scanf("%d", &ch);
        if(ch == 1){
        reflectX(p1);
        reflectX(p2);
        reflectX(p3);
        }
        else if(ch == 2){
        reflectY(p1);
        reflectY(p2);
        reflectY(p3);
        }
        else{
        reflect(p1);
        reflect(p2);
        reflect(p3);
        }
        Triangle(p1, p2, p3);
        break;
case 5:
        printf("Enter shearing value in x direction\n");
        scanf("%f", &shx);
```

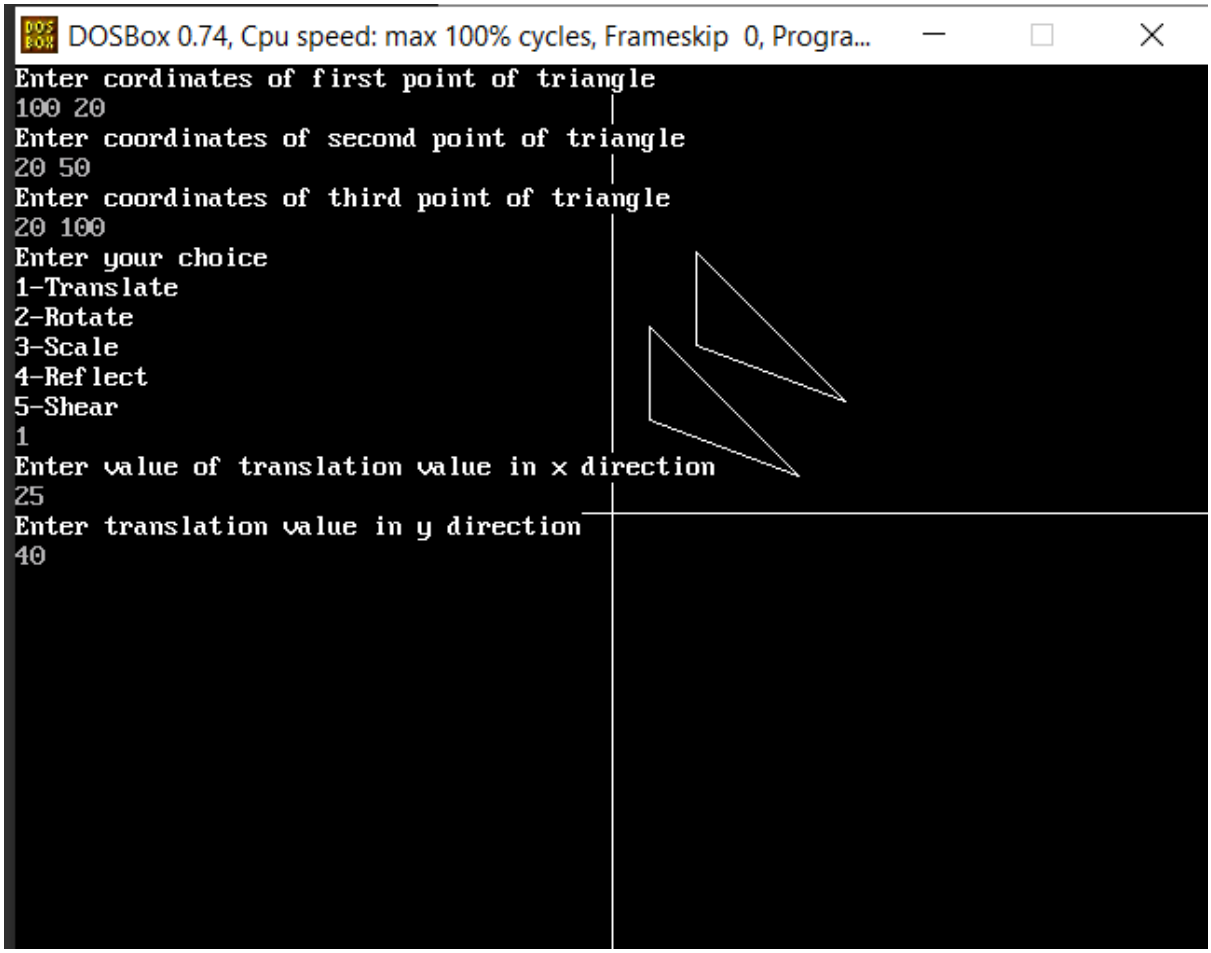
```
        printf("Enter shearing value in y direction\n");
        scanf("%f", &shy);
        shear(p1, shx, shy);
        shear(p2, shx, shy);
        shear(p3, shx, shy);
        Triangle(p1, p2, p3);
        break;
 }
 getch();
 closegraph();
}
void translate(float p[], float tx, float ty){
        int i, j;
        float mat[3][3];
        clearMat(mat);
 mat[0][0] = 1;
 mat[1][1] = 1;
 mat[2][2] = 1;
 mat[0][2] = tx;
 mat[1][2] = ty;
 multiply(mat, p);
}
void rotate(float p[], float theta){
        int i, j;
        float mat[3][3];
        clearMat(mat);
 mat[0][0] = cos(theta);
 mat[0][1] = -sin(theta);
 mat[1][0] = sin(theta);
```

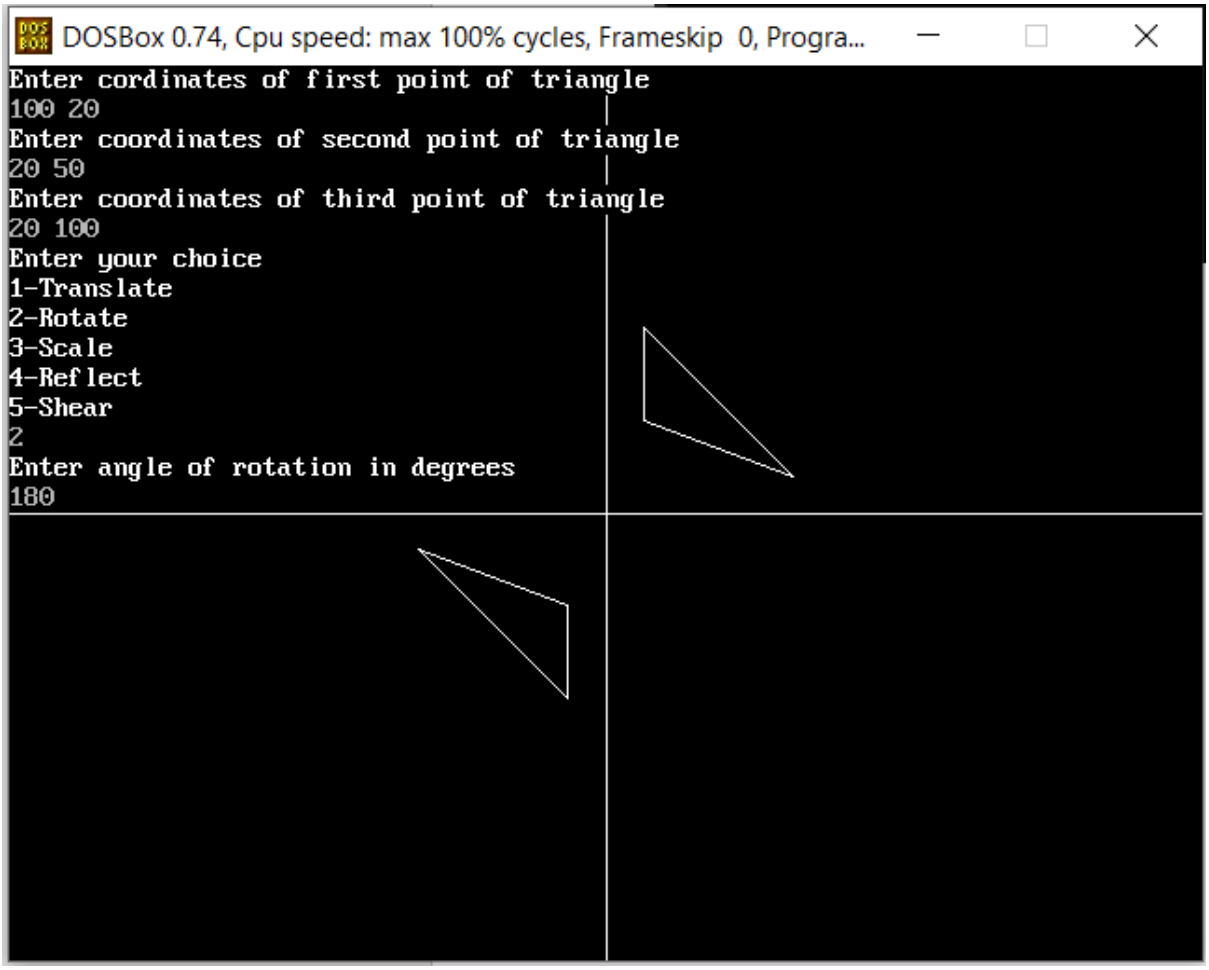
```
 mat[1][1] = cos(theta);
 mat[2][2] = 1;
 multiply(mat, p);
}
void scale(float p[3], float sx, float sy){
        int i, j;
        float mat[3][3];
        clearMat(mat);
 mat[0][0] = sx;
 mat[1][1] = sy;
 mat[2][2] = 1;
 multiply(mat, p);
}
void reflectX(float p[3]){
        float mat[3][3];
        clearMat(mat);
 mat[0][0] = 1;
 mat[1][1] = -1;
 mat[2][2] = 1;
 multiply(mat, p);
}
void reflectY(float p[3]){
        float mat[3][3];
        clearMat(mat);
 mat[0][0] = -1;
 mat[1][1] = 1;
 mat[2][2] = 1;
 multiply(mat, p);
}
```

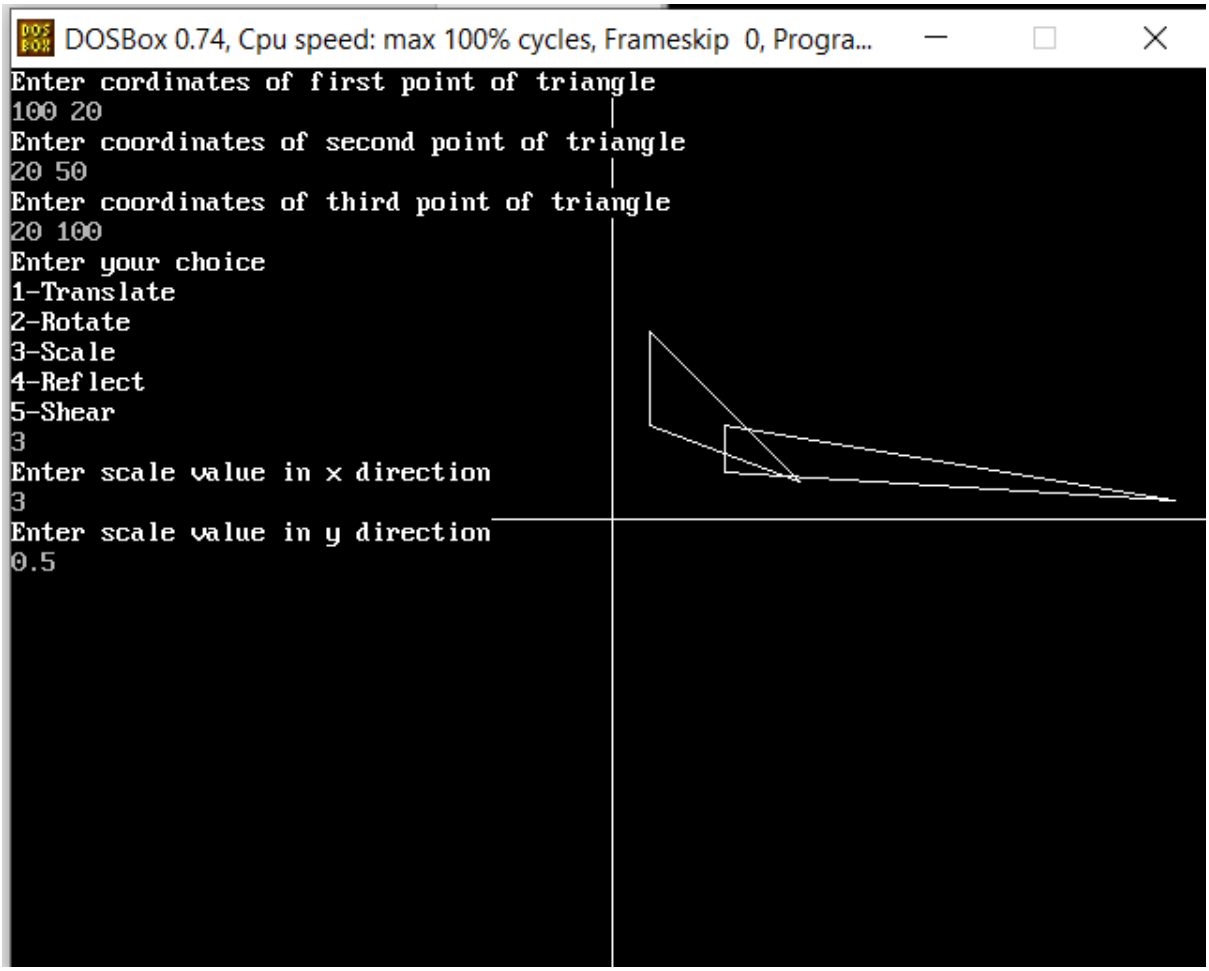
```
void reflect(float p[3]){
        float mat[3][3];
        clearMat(mat);
 mat[0][0] = -1;
 mat[1][1] = -1;
 mat[2][2] = 1;
 multiply(mat, p);
}
void shear(float p[3], float shx, float shy){
        float mat[3][3];
        clearMat(mat);
 mat[0][0] = 1; mat[1][1] = 1; mat[2][2] = 1;
 mat[1][0] = shx;
 mat[0][1] = shy;
 multiply(mat, p);
}
void multiply(float mat[3][3], float pt[3]){
        int pt0 = pt[0], pt1 = pt[1], pt2 = pt[2];
 pt[0] = pt0*mat[0][0] + pt1*mat[0][1] + pt2*mat[0][2];
 pt[1] = pt0*mat[1][0] + pt1*mat[1][1] + pt2*mat[1][2];
 pt[2] = pt0*mat[2][0] + pt1*mat[2][1] + pt2*mat[2][2];
}
void Triangle(float p1[], float p2[], float p3[]){
 line(320+p1[0], 240-p1[1], 320+p2[0], 240-p2[1]);
 line(320+p2[0], 240-p2[1], 320+p3[0], 240-p3[1]);
 line(320+p3[0], 240-p3[1], 320+p1[0], 240-p1[1]);
}
void clearMat(float mat[3][3]){
 int i, j;
```

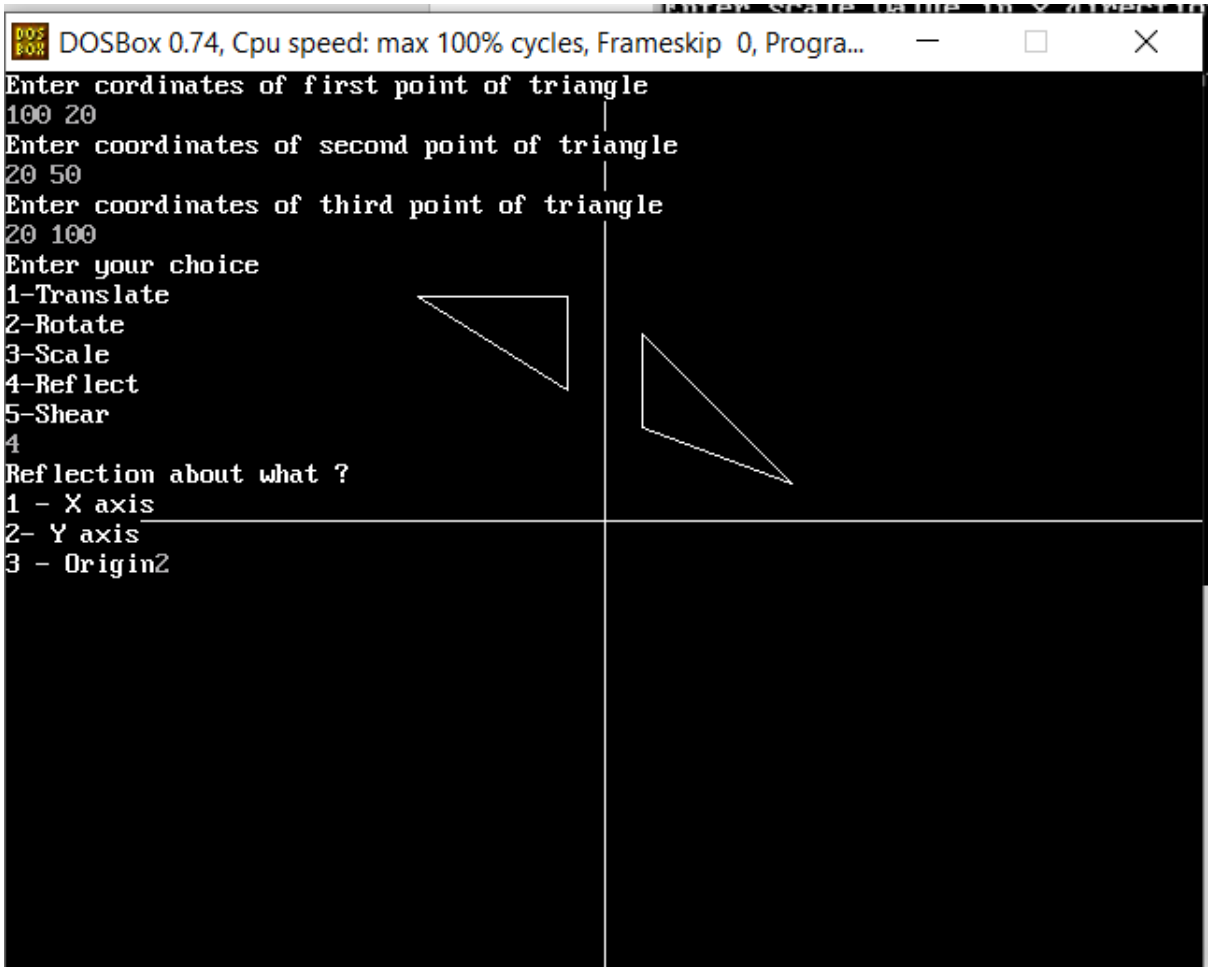
```
 for(i = 0; i < 3; i++)
        for(j = 0; j < 3; j++)
              mat[i][j] = 0;
}
```

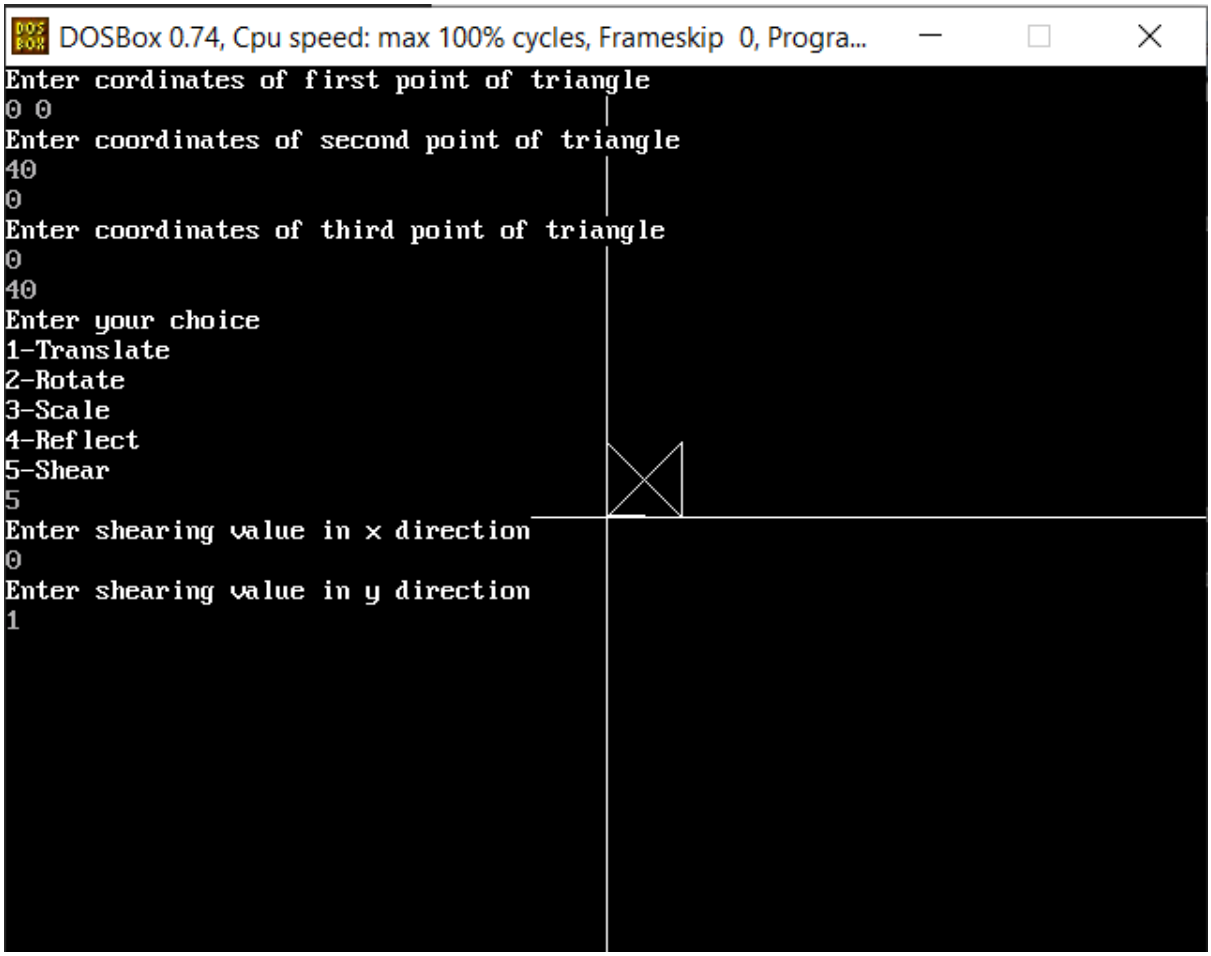
OUTPUT:




DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
Enter cordinates of first point of triangle
100 20
Enter coordinates of second point of triangle
20 50
Enter coordinates of third point of triangle
20 100
Enter your choice
1-Translate
2-Rotate
3-Scale
4-Reflect
5-Shear
2
Enter angle of rotation in degrees
180


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
Enter cordinates of first point of triangle
100 20
Enter coordinates of second point of triangle
20 50
Enter coordinates of third point of triangle
20 100
Enter your choice
1-Translate
2-Rotate
3-Scale
4-Reflect
5-Shear
3
Enter scale value in x direction
3
Enter scale value in y direction
0.5


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
Enter cordinates of first point of triangle
100 20
Enter coordinates of second point of triangle
20 50
Enter coordinates of third point of triangle
20 100
Enter your choice
1-Translate
2-Rotate
3-Scale
4-Reflect
5-Shear
4
Reflection about what ?
1 - X axis
2- Y axis
3 - Origin2


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
Enter cordinates of first point of triangle
0 0
Enter coordinates of second point of triangle
40
0
Enter coordinates of third point of triangle
0
40
Enter your choice
1-Translate
2-Rotate
3-Scale
4-Reflect
5-Shear
5
Enter shearing value in x direction
0
Enter shearing value in y direction
1

Harsh Kashiwal
2003085
C21

Page No.
Date

Experiment 7

Aim: Implement Liang Barsky line clipping.

Theory:

Algorithm:

Step 1: Take $x_{min}, x_{max}, y_{min}, y_{max}, x_1, y_1, x_2, y_2$ as input from the user and set $t_1 = 0$ $t_2 = 1$

Step 2: Calculate $\Delta x = x_2 - x_1$, $\Delta y = y_2 - y_1$

$p_1 = -\Delta x$, $q_1 = x_1 - x_{min}$

$p_2 = \Delta x$, $q_2 = x_{max} - x_1$

$p_3 = -\Delta y$, $q_3 = y_1 - y_{min}$

$p_4 = \Delta y$, $q_4 = y_{max} - y_1$

Step 3: For each value of k from 1 to 4, perform the following check.

If $p_k = 0$ and $q_k < 0$ end the algorithm.

Step 4: For each value of k from 1 to 4 execute the following:

If $p_k < 0$ and $q_k/p_k > t_1$

then set $t_1 = q_k/p_k$

otherwise if $q_k/p_k < t_2$ then set $t_2 = q_k/p_k$.

Step 5: If $t_1 < t_2$, calculate $(x_i, y_i) = (x_1 + t_1\Delta x, y_1 + t_1\Delta y)$

Step 6: Draw the line from (x_i, y_i) to (x_f, y_f) where $(x_f, y_f) = (x_1 + t_2\Delta x, y_1 + t_2\Delta y)$.

PROGRAM:

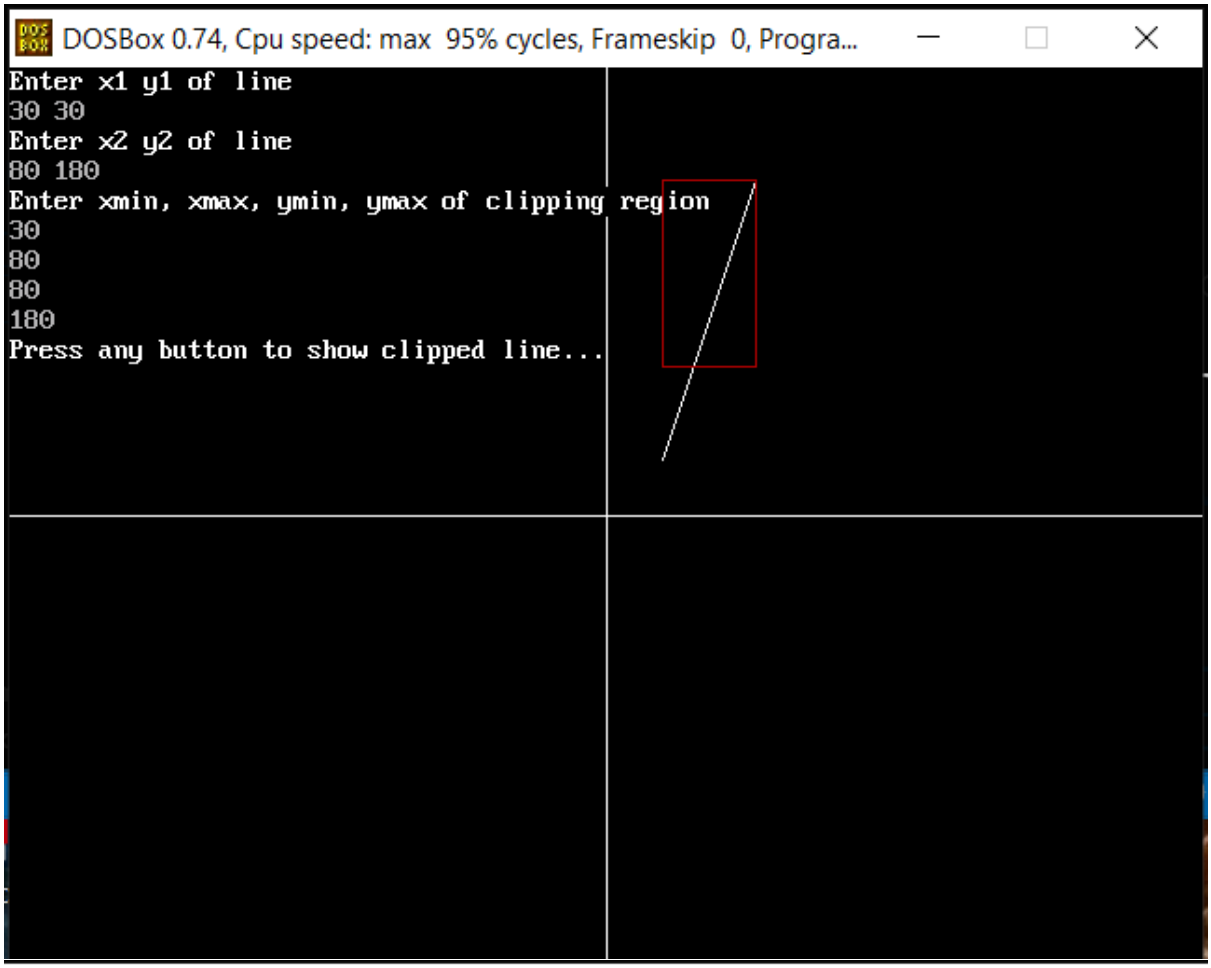
Code:

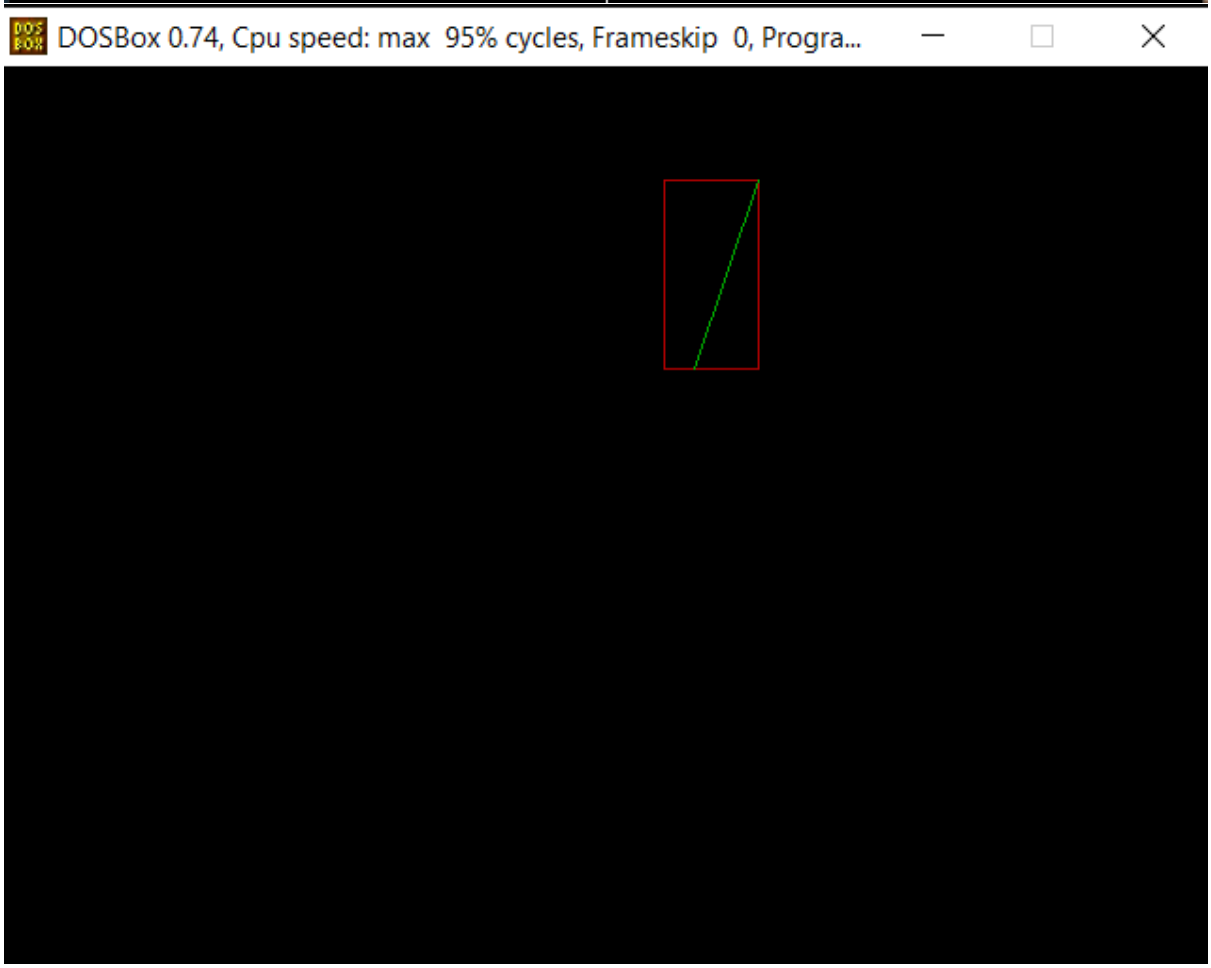
```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
void ClippedLine(int, int, int, int, int, int, int, int, int);
void Line(int, int, int, int);
void main()
{
   int x1, y1, x2, y2, xmin, xmax, ymin, ymax;
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
   line(320, 0, 320, 480);
   line(0, 240, 640, 240);
   printf("Enter x1 y1 of line\n");
   scanf("%d %d", &x1, &y1);
   printf("Enter x2 y2 of line\n");
   scanf("%d %d", &x2, &y2);
   Line(x1, y1, x2, y2);
   printf("Enter xmin, xmax, ymin, ymax of clipping region\n");
   scanf("%d %d %d %d", &xmin, &xmax, &ymin, &ymax);
   setcolor(RED);
   rectangle(320 + xmin, 240 - ymin, 320 + xmax, 240 - ymax);
   printf("Press any button to show clipped line...\n");
   getch();
   cleardevice();
   rectangle(320 + xmin, 240 - ymin, 320 + xmax, 240 - ymax);
```

```
    ClippedLine(x1, y1, x2, y2, xmin, ymin, xmax, ymax,
                GREEN);
    getch();
    getch();
    closegraph();
}
void ClippedLine(int x1, int y1, int x2, int y2, int xmin, int ymin, int xmax, int
ymax, int col)
{
    float t1, t2;
    int dx = x2 - x1, dy = y2 - y1, p[4], q[4], k;
    p[0] = -dx;
    p[1] = dx;
    p[2] = -dy;
    p[3] = dy;
    q[0] = x1 - xmin;
    q[1] = xmax - x1;
    q[2] = y1 - ymin;
    q[3] = ymax - y1;
    for (k = 0; k < 4; k++)
       if (p[k] == 0 && q[k] < 0)
          return;
    t1 = 0;
    t2 = 1;
    for (k = 0; k < 4; k++)
    {
       if (p[k] < 0 && (float)q[k] / p[k] > t1)
          t1 = (float)q[k] / p[k];
```

```
        else if (p[k] > 0 && (float)q[k] / p[k] < t2)
            t2 = (float)q[k] / p[k];
    }
    if (t1 < t2)
    {
        setcolor(col);
        Line(x1 + t1 * dx, y1 + t1 * dy, x1 + t2 * dx, y1 + t2 * dy);
    }
}
void Line(int x1, int y1, int x2, int y2)
{
    line(320 + x1, 240 - y1, 320 + x2, 240 - y2);
}
```

OUTPUT:


DOSBox 0.74, Cpu speed: max 95% cycles, Frameskip 0, Progra...
Enter x1 y1 of line
30 30
Enter x2 y2 of line
80 180
Enter xmin, xmax, ymin, ymax of clipping region
30
80
80
180
Press any button to show clipped line...


DOSBox 0.74, Cpu speed: max 95% cycles, Frameskip 0, Progra...

Harsh Kashiwal
2003085
C21

Page No.
Date

Experiment 8

Aim: Implement Sutherland Hodgeman polygon clipping method in C.

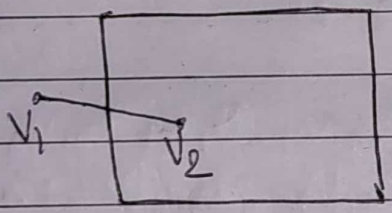
Theory:

Algorithm:

Step 1: Take input, vertices and edges of clip window. Also, create a list to store vertices.

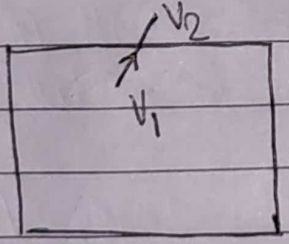
Step 2: Traverse through the edges of the polygon in order check which condition each edge falls into.

1) Outside → Inside.



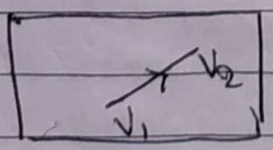
In this case, add intersection point & destination point to the list.

2) Inside → Outside.



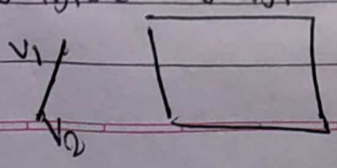
In this case, only add intersection point to the list.

3) Inside → Inside.



In this case, add destination point to the list.

4) Outside → Outside



Don't add

Page No.
Date

Step 3: Repeat step 2 for each edge of the clipping window, that is left, right, top and bottom, after clearing the list and considering the list as the new polygon.

Step 4: Traverse the vertices in the list in order and draw each edge.

PROGRAM:

Code:

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>

float pts[20];
int count = 0;
int xmin, ymin, xmax, ymax;
void intersectX(float, float, float, float, float), intersectY(float, float, float, float, float);
void clipLeft(int, int, int, int), clipRight(int, int, int, int), clipTop(int, int, int, int), clipBottom(int, int, int, int);

void main()
{
	int gd = DETECT, gm;
	float tri[20]; //Original points of the triangle
	int i, n;
	initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
	printf("Enter the coordinates of your triangle as x1 y1 x2 y2 x3 y3\n");
	scanf("%f %f %f %f %f %f", &tri[0], &tri[1], &tri[2], &tri[3], &tri[4], &tri[5], &tri[6]);
	tri[6] = tri[0];
	tri[7] = tri[1];
	for(i = 0; i < 6; i+=2)
		line(tri[i], tri[i+1], tri[i+2], tri[i+3]);
```

```
	printf("Enter the coordinates of the clipping window as xmin ymin xmax ymax\n");
	scanf("%d %d %d %d", &xmin, &ymin, &xmax, &ymax);

	setcolor(RED);
	rectangle(xmin, ymin, xmax, ymax);

	setcolor(WHITE);

	printf("Press any button to show the clipped triangle...");
	getch();

	cleardevice();
	setcolor(RED);
	rectangle(xmin, ymin, xmax, ymax);

	count = 0;
	for(i = 0; i < 6; i+=2)
		clipLeft(tri[i], tri[i+1], tri[i+2], tri[i+3]);

	n = count;
	count = 0;
	for(i = 0; i < n; i++)
		tri[i] = pts[i];
	tri[i] = pts[0];
	tri[i+1] = pts[1];
```

```
for(i = 0; i < n; i+=2)
        clipBottom(tri[i], tri[i+1], tri[i+2], tri[i+3]);

n = count;
count = 0;
for(i = 0; i < n; i++)
        tri[i] = pts[i];
tri[i] = pts[0];
tri[i+1] = pts[1];

for(i = 0; i < n; i+=2)
        clipRight(tri[i], tri[i+1], tri[i+2], tri[i+3]);

n = count;
count = 0;
for(i = 0; i < n; i++)
        tri[i] = pts[i];
tri[i] = pts[0];
tri[i+1] = pts[1];

for(i = 0; i < n; i+=2)
        clipTop(tri[i], tri[i+1], tri[i+2], tri[i+3]);

n = count;
count = 0;
for(i = 0; i < n; i++)
        tri[i] = pts[i];
```

```
        tri[i] = pts[0];
        tri[i+1] = pts[1];

        setcolor(GREEN);
        for(i = 0; i < n; i+=2)
                line(tri[i], tri[i+1], tri[i+2], tri[i+3]);

        getch();
        getch();
}

void clipLeft(int x1, int y1, int x2, int y2)
{
        int i, initCount = count;
        if(x1 <= xmin && x2 >= xmin) //Outside to inside
        {
                intersectX(x1, y1, x2, y2, xmin);
                pts[count++] = x2;
                pts[count++] = y2;
        }
        else if(x1 >= xmin && x2 <= xmin) //Inside to outside
        {
                intersectX(x1, y1, x2, y2, xmin);
        }
        else if(x1 >= xmin && x2 >= xmin) //Inside to inside
        {
                pts[count++] = x2;
```

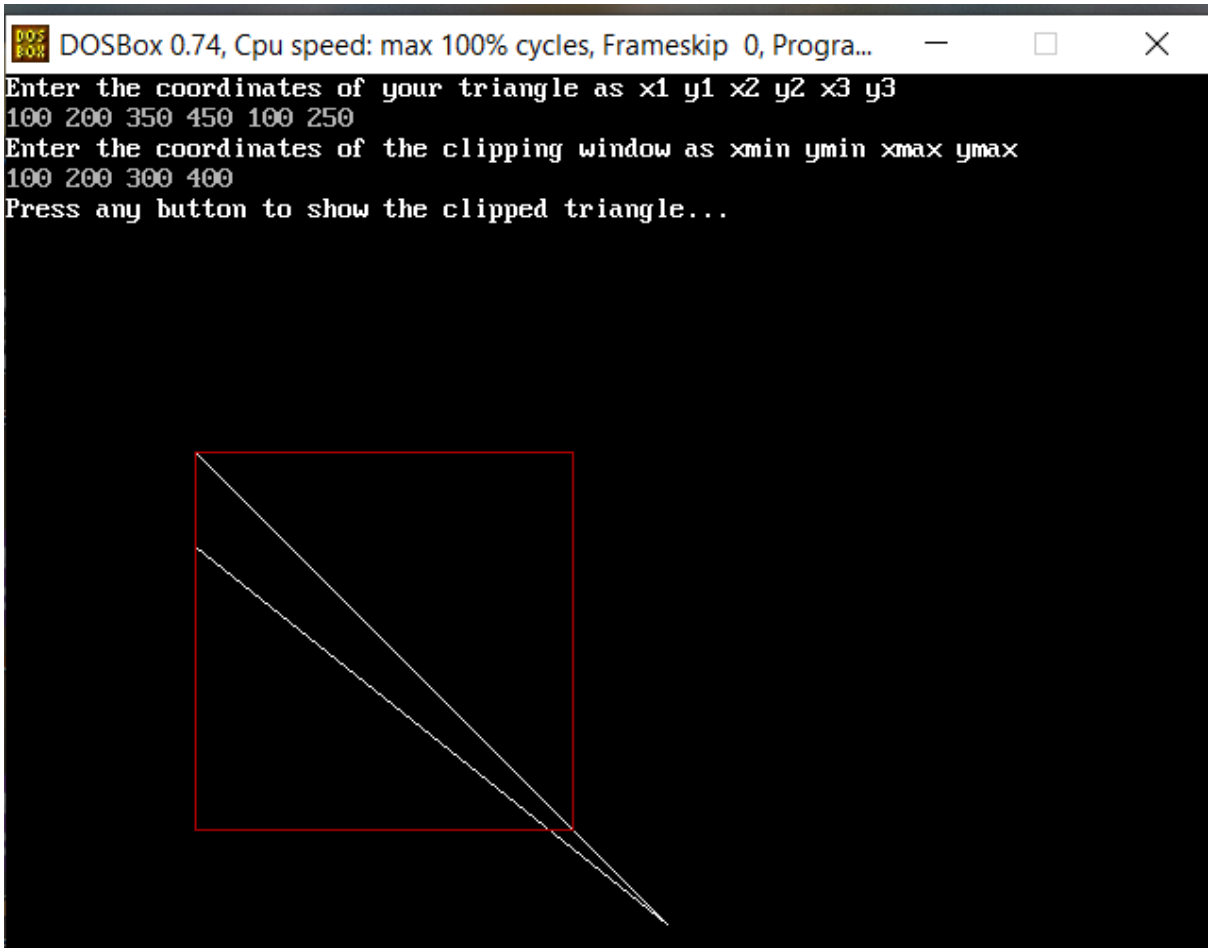
```
            pts[count++] = y2;
        }
        //else outside to outside, don't save anything
}

void clipRight(int x1, int y1, int x2, int y2)
{
        int i, initCount = count;
        if(x1 >= xmax && x2 <= xmax) //Outside to inside
        {
            intersectX(x1, y1, x2, y2, xmax);
            pts[count++] = x2;
            pts[count++] = y2;
        }
        else if(x1 <= xmax && x2 >= xmax) //Inside to outside
        {
            intersectX(x1, y1, x2, y2, xmax);
        }
        else if(x1 <= xmax && x2 <= xmax) //Inside to inside
        {
            pts[count++] = x2;
            pts[count++] = y2;
        }
        //else outside to outside, don't save anything
}

void clipTop(int x1, int y1, int x2, int y2)
```

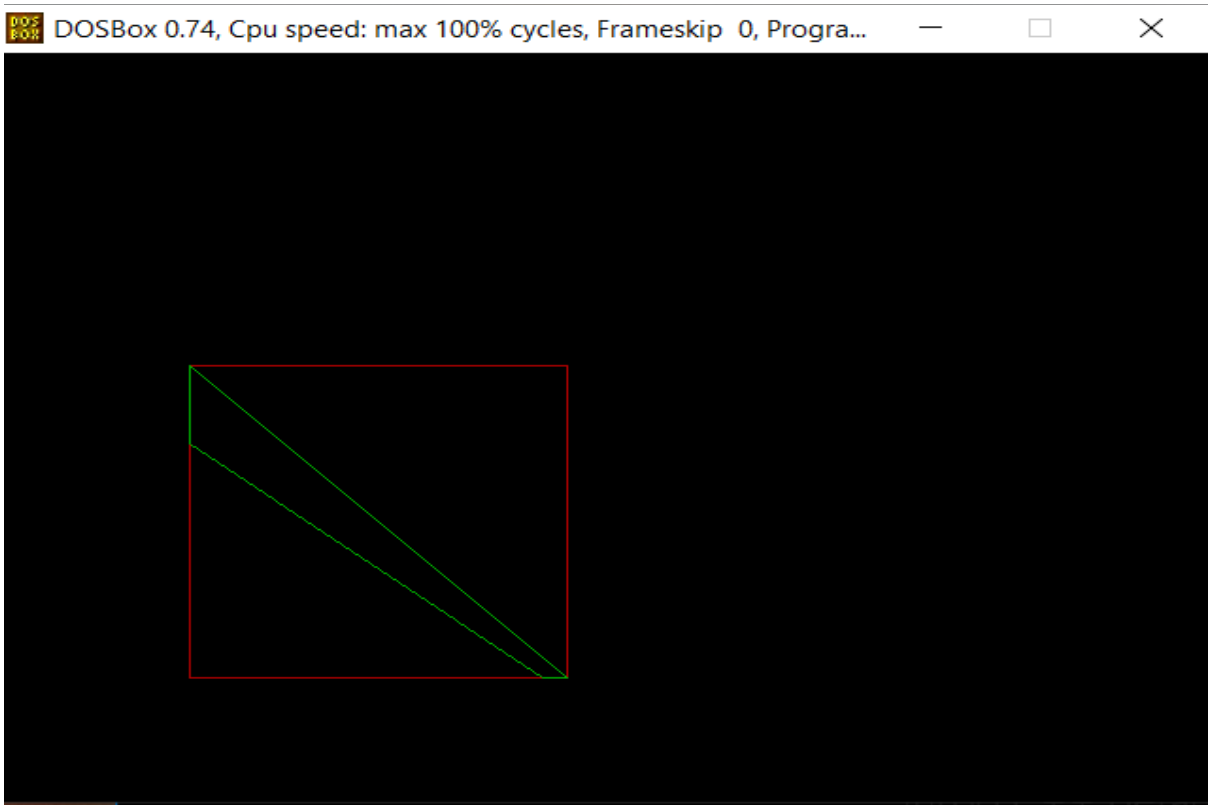
```
{
	int i, initCount = count;
	if(y1 <= ymin && y2 >= ymin) //Outside to inside
	{
		intersectY(x1, y1, x2, y2, ymin);
		pts[count++] = x2;
		pts[count++] = y2;
	}
	else if(y1 >= ymin && y2 <= ymin) //Inside to outside
	{
		intersectY(x1, y1, x2, y2, ymin);
	}
	else if(y1 >= ymin && y2 >= ymin) //Inside to inside
	{
		pts[count++] = x2;
		pts[count++] = y2;
	}
	//else outside to outside, don't save anything
}

void clipBottom(int x1, int y1, int x2, int y2)
{
	int i, initCount = count;
	if(y1 >= ymax && y2 <= ymax) //Outside to inside
	{
		intersectY(x1, y1, x2, y2, ymax);
		pts[count++] = x2;
```

```
            pts[count++] = y2;
        }
        else if(y1 <= ymax && y2 >= ymax) //Inside to outside
        {
            intersectY(x1, y1, x2, y2, ymax);
        }
        else if(y1 <= ymax && y2 <= ymax) //Inside to inside
        {
            pts[count++] = x2;
            pts[count++] = y2;
        }
        //else outside to outside, don't save anything
}

void intersectX(float x1, float y1, float x2, float y2, float x)
{
        //y = mx + c
        //=> y1 = mx1 + c
        //=> c = y1-mx1
        float m = (y2-y1)/(x2-x1), c = y1-m*x1;
        float y = m*x+c;
        pts[count++] = x;
        pts[count++] = y;
}

void intersectY(float x1, float y1, float x2, float y2, float y)
{
```

```
        float m = (y2-y1)/(x2-x1), c = y1-m*x1;
        float x = (y-c)/m;
        if(x2 == x1) //If slope is INF
        {
                x = x1;
        }
        pts[count++]=x;
        pts[count++]=y;
}
```

OUTPUT:


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
Enter the coordinates of your triangle as x1 y1 x2 y2 x3 y3
100 200 350 450 100 250
Enter the coordinates of the clipping window as xmin ymin xmax ymax
100 200 300 400
Press any button to show the clipped triangle...


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

Harsh Kasliwal
2009085
(2)

Page No.
Date

Experiment 9

Aim: Implement Bezier curve.

Theory: A Bezier curve is a parametric curve which can be controlled using any number of control points. It is extensively used in computer graphics and engineering.

Algorithm:

Step 1: Take the control points coordinates as input (P_{0x}, P_{1x} ... P_{0y}, P_{1y} ...)

step 2: Set $t=0$

step 3: Set $i=0$, $P_x=0$, $P_y=0$

step 4: Set $P_x = P_x + P_{ix} * n_{C_i} * t^i * (1-t)^{(n-i)}$

Set $P_y = P_y + P_{iy} * n_{C_i} * t^i * (1-t)^{(n-i)}$.

step 5: Repeat step 4 until $i \leq n$ that is repeat step 4 for all control points.

step 6: Put pixel at P_x, P_y.

step 7: Set $t = t + 0.001$ [or whichever value is wanted. Lower the value, more points of the curve will be plotted].

step 8: If $t \leq 1$, goto step 5.

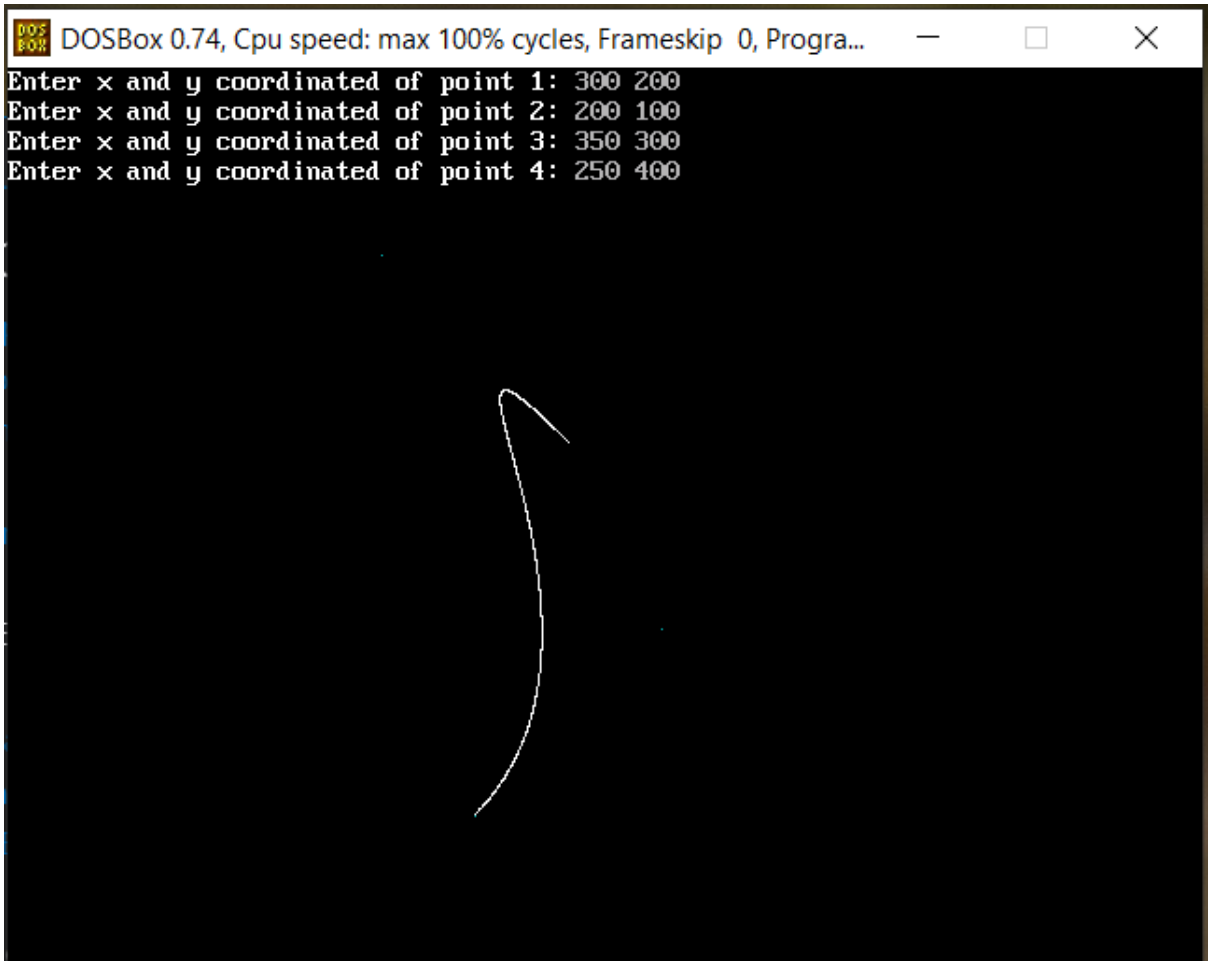
The Bezier curve has been plotted !!.

PROGRAM:

Code:

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<math.h>
void main(){
   int x[4],y[4],i;
   double puty,putx,t;
   int gd=DETECT,gm;
   initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
   for ( i = 0; i < 4; i++)
   {
      printf("Enter x and y coordinated of point %d: ",i+1);
      scanf("%d%d",&x[i],&y[i]);
      putpixel(x[i],y[i],3);


   }
   for(t=0.0;t<=1.0;t=t+0.001){
      putx=pow(1-t,3)*x[0]+ 3*t*pow(1-t,2)*x[1]+ 3*t*t*pow(1-t,1)*x[2]+
pow(t,3)*x[3];
      puty=pow(1-t,3)*y[0]+ 3*t*pow(1-t,2)*y[1]+ 3*t*t*pow(1-t,1)*y[2]+
pow(t,3)*y[3];
      putpixel(putx,puty,WHITE);
   }
   getch();
   closegraph();
   }
```

OUTPUT:


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
Enter x and y coordinated of point 1: 300 200
Enter x and y coordinated of point 2: 200 100
Enter x and y coordinated of point 3: 350 300
Enter x and y coordinated of point 4: 250 400

Harsh Kaliwal
2002085
C24

Page No.
Date

Experiment 10

Aim: Implement koch curve for fractal generation.

Theory: The curve is a type of fractal curve in which the next iteration of the curve is formed by adding an outward bend to each line segment in the previous iteration.

Algorithm:

```
drawKoch(x1, y1, x2, y2, n):
    if n is 0:
        drawLine(x1, y1, x2, y2)
    else:
        calculate x3, y3, x4, y4, θ, L, x5, y5 as
        x3 = (1*x2 + 2*x1)/3 , y3 = (1*y2 + 2*y1)/3
        x4 = (2*x2 + 1*x1)/3 , y4 = (2*y2 + 1*y1)/3
        θ = tan^-1((y2 - y1)/(x2 - x1))
        L = sqrt((x2 - x1)^2 + (y2 - y1)^2)
        x5 = x3 + (1/3) L cos(θ + π/3)

        y5 = y3 + (1/3) L sin(θ + π/3)

        drawKoch(x1, y1, x3, y3, n-1)
        drawKoch(x3, y3, x5, y5, n-1)
        drawKoch(x5, y5, x4, y4, n-1)
        drawKoch(x4, y4, x2, y2, n-1)
```

Take number of iterations (n) as input from the user and call drawKoch(100, 200, 300, 200, n).

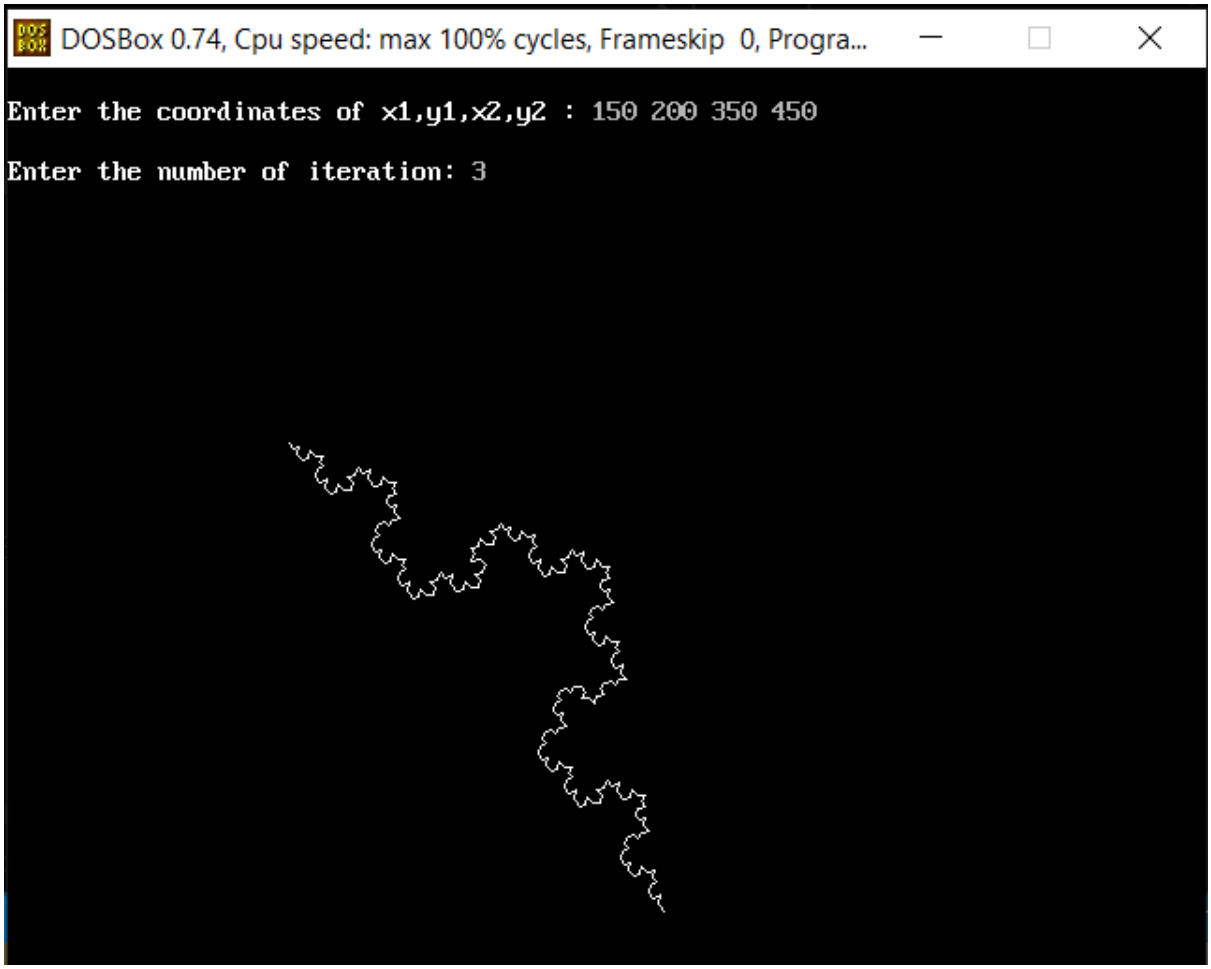
PROGRAM:

Code:

```
#include<graphics.h>
#include<conio.h>
#include<math.h>
void koch(int x1, int y1, int x2, int y2, int iteration)
{
 float angle = 60*M_PI/180;
 int x3 = (2*x1+x2)/3;
 int y3 = (2*y1+y2)/3;
 int x4 = (x1+2*x2)/3;
 int y4 = (y1+2*y2)/3;
 int x = x3 + (x4-x3)*cos(angle)+(y4-y3)*sin(angle);
 int y = y3 - (x4-x3)*sin(angle)+(y4-y3)*cos(angle);
 if(iteration > 0)
 {
 koch(x1, y1, x3, y3, iteration-1);
 koch(x3, y3, x, y, iteration-1);
 koch(x, y, x4, y4, iteration-1);
 koch(x4, y4, x2, y2, iteration-1);
 }
 else
 {
 line(x1, y1, x3, y3);
 line(x3, y3, x, y);
 line(x, y, x4, y4);
 line(x4, y4, x2, y2);
```

```
 }
}
int main(void)
{
 int gd = DETECT, gm;
 int x1, y1 , x2, y2, n;
 initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
 printf("\nEnter the coordinates of x1,y1,x2,y2 : ");
 scanf("%d%d%d%d",&x1,&y1,&x2,&y2 );
 printf("\nEnter the number of iteration: ");
 scanf("%d",&n);
 koch(x1, y1, x2, y2, n);
 getch();
 return 0;
}
```

OUTPUT:


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
Enter the coordinates of x1,y1,x2,y2 : 150 200 350 450
Enter the number of iteration: 3

AIR HOCKEY GAME

Garima Kakwani-2003078

Harsh Kasliwal-2003085

Viren Kalra-2003079

PROGRAM STATEMENT:

Write a program to build a responsive Air Hockey Game in C.

THEORY:

We built a 2-player virtual air hockey game. Our goal was to implement an exciting and easy-to play game without the burdensome physical setup. The players are able to control the mallets.

Air hockey works as follows, Control keys for moving two colored rectangles i.e., mallet is ‘a’, ‘d’, ‘j’ and l’; ‘a’ and ‘d’ is to move upper mallet left and right respectively and ‘j’ and ‘l’ is used to move lower mallet left and right respectively. The puck bounces off the two sides of table(screen), which are raised to form walls by dotted line. The other two walls of the table(screen) are two goals, one defended by each player. The objective of the game is to hit the puck into the opponent’s goal. When a goal is scored, the scoring player wins a point and the game will over and winner will be displayed on the screen. User as press enter to play the game again.

In this project this, we have included two main hearder files, *graphics.h* and stdio.h. Graphics.h header file provides access to a simple graphics library that makes it possible to draw lines, rectangles, arcs, circle, display text in different fonts, change colors, strings and many more on a graphical window.

FUNCTIONS USED:

1. **Initgraphics():**

   initgraph **initializes the graphics system by loading a graphics driver from disk** (or validating a registered driver), and putting the system into graphics mode.

2. **Line ():**

   line Function Draws Line From (x1, y1) to (x2, y2).
   x1 - X Co-ordinate of First Point
   y1 - Y Co-ordinate of First Point
   x2 - X Co-ordinate of Second Point
   y2 - Y Co-ordinate of Second Point
   For example, in our program we have drawn a line in the Middle of the Screen

   line (0, 240, 640, 240);

3. **setcolor()**

   setcolor() function which is used to set the current drawing color to the new color.
   Example:

   setcolor(RED);
   sprintf(txt,"AIR HOCKEY!");
   This will print AIR HOCKEY! In RED color.

4. **Outtextxy()**

   outtextxy() function which displays the text or string at a specified point (x, y) on the screen.
   Syntax: *void outtextxy(int x, int y, char *string);*
   where, x, y are coordinates of the point and, third argument contains the address of string to be displayed.
   Example:

   sprintf(txt,"Press Enter to play again...");
   outtextxy(150,midy+40,txt);

   This display the value store in the string ‘txt’ i.e Press Enter to play again…
   At x=150 and y= midy+40
   Where midy is middle point of y-cordinate.

5. **getmaxy():**
   getmaxy returns the maximum y value for the current graphics driver and mode. And i.e approx. 480

6. **getmaxx():**

   getmaxx returns the maximum x value for the current graphics driver and mode

   and i.e approx. 640

7. **setlinestyle():** setlinestyle() function is used to change the pattern and thickness of the font of the text.
   Syntax:

   void setlinestyle(int linestyle, unsigned upattern, int thickness);

   Example:

   ```
   setlinestyle(1,1,3);
    line(1,1,1,maxy);
   ```

   1 will print Dotted line of thickness 3.

Name	Value	Description
SOLID_LINE	0	Solid line
DOTTED_LINE	1	Dotted line
CENTER_LINE	2	Centered line
DASHED_LINE	3	Dashed line
USERBIT_LINE	4	User-defined line style

Name	Value	Description
NORM_WIDTH	1	1 pixel wide
THICK_WIDTH	3	3 pixels wide

8. **settextstyle():**
   setlinestyle() function is used to change the way the text appears, using it we can modify the size of text, direction of text and change the font of text.

   Syntax:

   void settextstyle(int font, int direction, int font_size);

where,

font argument specifies the font of text, Direction can be HORIZ_DIR (Left to right) or VERT_DIR (Bottom to top).

Example:

settextstyle(7,0,5);

7 is int value for TRIPLEX_SCR_FONT

0 indicates that text will be print in Horizontal direction

5 is the font size

9. **circle ():**
   circle (*x,y,r*) function which draws a circle with centre at (x, y) and r is radius. Example:
10. **rectangle ():**
    rectangle(*int left, int top, int right, int bottom*) function is used to draw a rectangle. Coordinates of left top and right bottom corner are required to draw the rectangle.

11. **sprintf():**sprint stands for “String print”. Instead of printing on console, it store output on char buffer which are specified in sprintf.

    Example:

    char txt[50];

    sprintf(txt,"Player-2 Win");

    outtextxy(midx-120,midy-10,txt);

    String txt will store the message instead of printing on the console.

12. **kbhit():** It is present in conio.h and used to determine if a key has been pressed or not.

13. **Cleardevice():** cleardevice() function in C clears the screen in graphics mode and sets the current position to (0,0).

14. **Closegraph():** closegraph function closes the graphics mode and deallocates all memory allocated by graphics system .

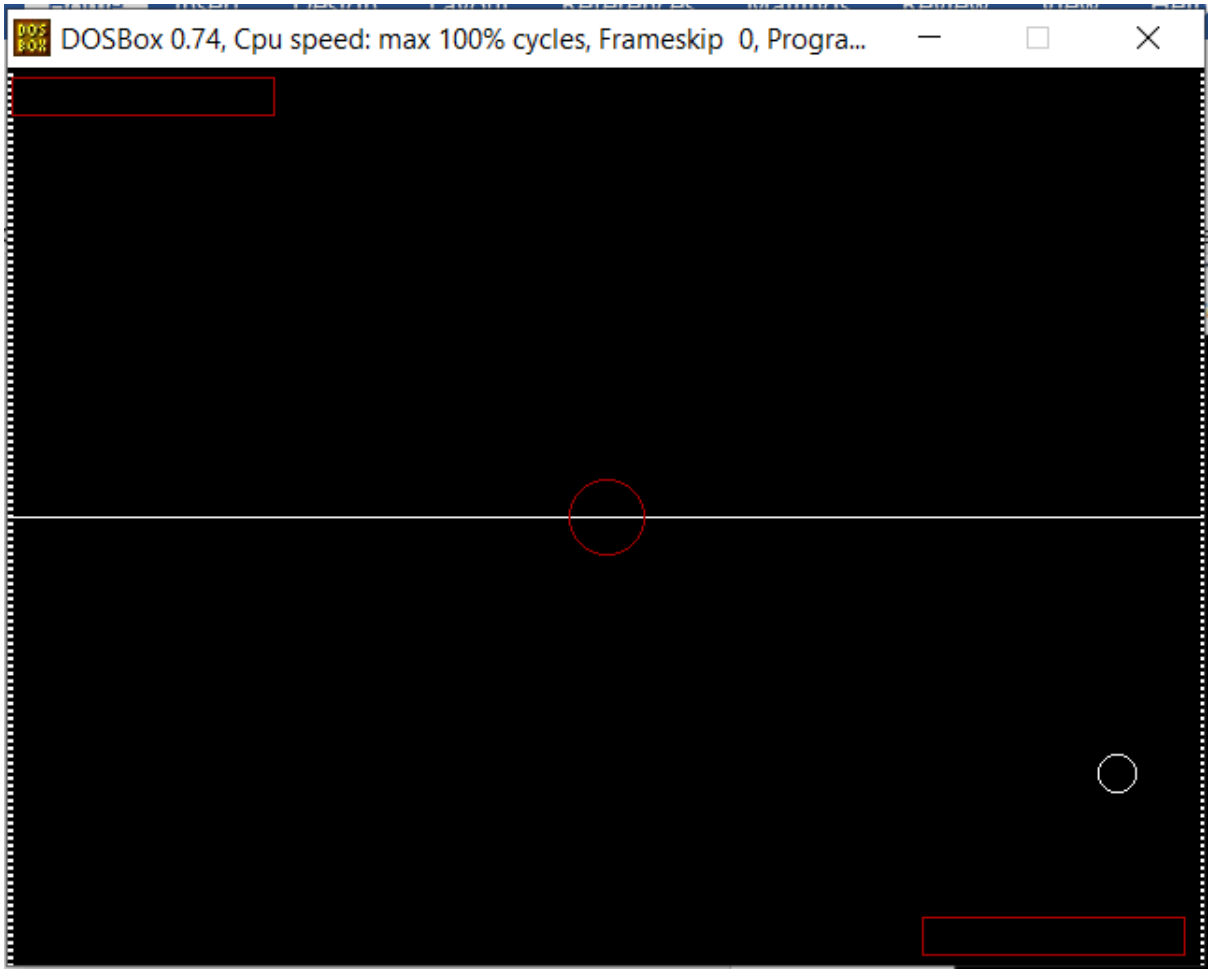
CODE:

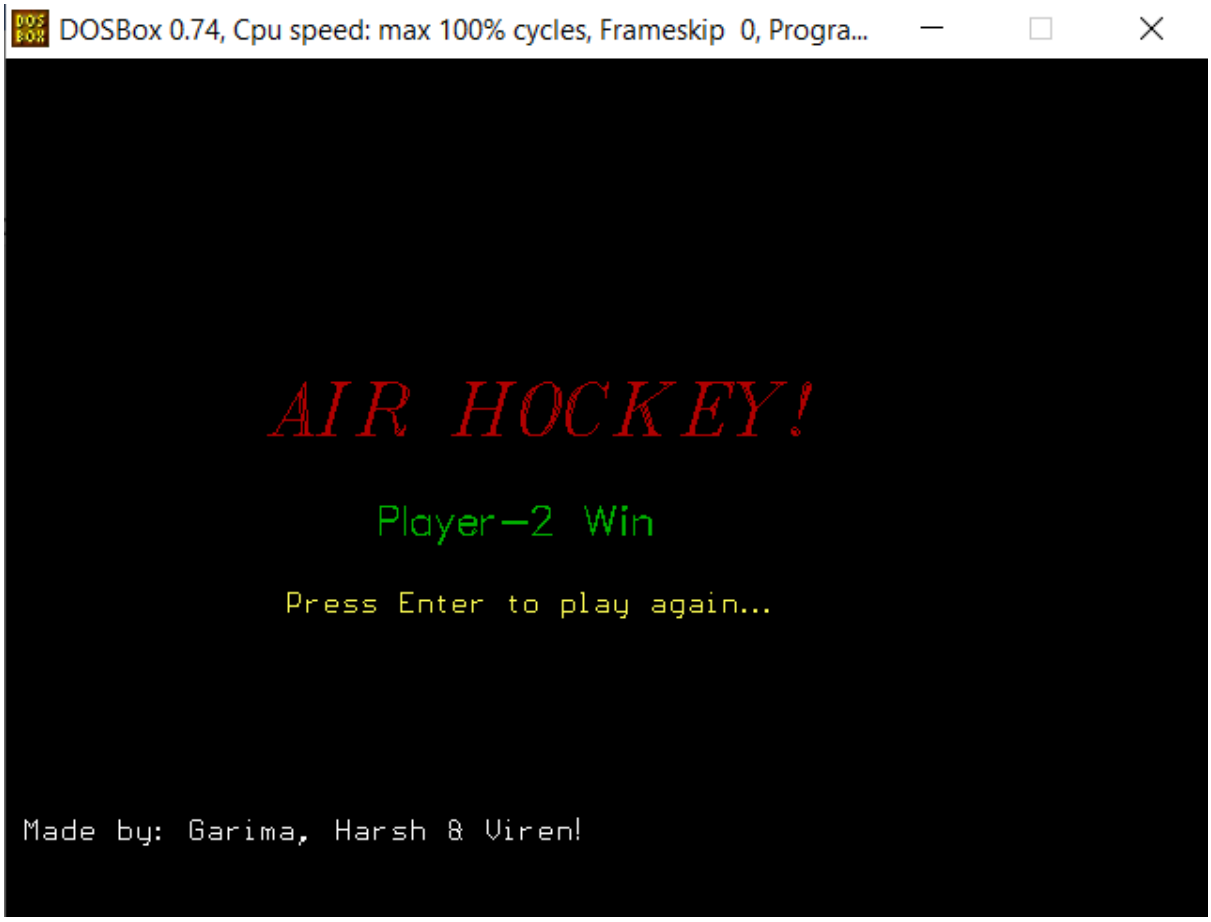
```
#include<graphics.h>
#include<stdlib.h>
#include<conio.h>
#include<stdio.h>
#include<dos.h>
void main(){
 int gd=DETECT,gm;
 int xb,yb,xr,yr,xR,yR,key,maxx,maxy,midx,midy,xspeed,yspeed,win;
 char txt[50];
 initgraph(&gd,&gm,"C:\\TurboC3\\BGI");
 Play_Again:
 maxx=getmaxx();
 maxy=getmaxy();
 midx=getmaxx()/2;
 midy=getmaxy()/2;
 xb=midx;
 yb=midy;
 xr=maxx-150;
 yr=maxy-25;
 xR=random(maxx);
 yR=5;
 xspeed=2;
 yspeed=2;
 setbkcolor(0);
 setcolor(GREEN);
 while(1)
 {
  if(kbhit())
  {
    key=getch();
    if(key==106)
      xr-=8;
    else if(key==108)
      xr+=8;
    else if(key==97)
      xR-=8;
    else if(key==100)
```

```
      xR+=8;
    else if(key==27)
      break;
  }
  xb+=xspeed;
  yb+=yspeed;
  setcolor(WHITE);
  line(0, 240, 640, 240);
  setcolor(RED);
  circle(320,240,20);
  setcolor(WHITE);
  setlinestyle(1,1,3);
  line(1,1,1,maxy);
  line(maxx,1,maxx,maxy);
  setcolor(WHITE);
  setlinestyle(0,1,0);
  circle(xb,yb,10);
  if(xb>(maxx-10)||xb<10)
    xspeed*=(1-2);
  if(yb<10)
  {
    win=1;
    break;
  }
  if(yb>(maxy-10))
  {
    win=2;
    break;
  }
  if((yr<(yb+10)&&(yr+20)>(yb+10))&&(xr<(xb-
10)&&(xr+140)>(xb+10)))
    yspeed*=(1-2);
  if(((yR+20)>(yb-10)&&yR<(yb-10))&&(xR<(xb-
10)&&(xR+140)>(xb+10)))
    yspeed*=(1-2);
  setcolor(RED);
  rectangle(xr,yr,xr+140,yr+20);
  rectangle(xR,yR,xR+140,yR+20);
  delay(10);
  cleardevice();
```

```
        }
        cleardevice();
        settextstyle(7,0,5);
        setcolor(RED);
        sprintf(txt,"AIR HOCKEY!");
        outtextxy(midx-180,midy-80,txt);
        settextstyle(3,0,3);
        setcolor(GREEN);
        if(win==2)
          sprintf(txt,"Player-1 Win");
        else
          sprintf(txt,"Player-2 Win");
        outtextxy(midx-120,midy-10,txt);
        settextstyle(2,0,6);
        setcolor(YELLOW);
        sprintf(txt,"Press Enter to play again...");
        outtextxy(150,midy+40,txt);
        setcolor(WHITE);
        outtextxy(10,400,"Made by: Garima, Harsh & Viren!");
        if((getch()==13))
        goto Play_Again;
        getch();
        closegraph();
}
```

OUTPUT:




DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Progra...
AIR HOCKEY!
Player-2 Win
Press Enter to play again...
Made by: Garima, Harsh & Viren!

2003085 · IC21

Computer Graphics:

i) Explain architecture of Raster Scan Display:

=> In raster scan the electron beam follows a fixed path. The Electron beam starts at top left corner of the screen and moves horizontally to the right. This defines a scan line. During the scan the intensity of the beam is modulated according to the pattern of the desired image along the line.

At the right corner of the screen, the beam becomes off and moved back to the left edge of the screen at the starting point on next line. This is shown as dotted line which is called as horizontal retrace. This way of scanning is continued to the bottomed right corner is reached. At this point, one scan is completed. The beam is then repositioned at the top left corner of the screen for starting another scan. This movement of beam from bottom right corner to top left corner is called as vertical retrace.

ii) Describe applications of computer graphics:

=> A) Virtual Reality:

With the help of computer graphics. It is possible to simulate various virtual environments which looks seamlessly similar to real world. Such environment helps to train the person without providing them with physical resources. Doctors can be trained to operate a patient without operating on the actual physical body.

B) Presentation graphics:

Another major application of CG is presentation graphics. It is used for producing diagrams, tables and charts for reports or presentations. It is generally used to summarize research or survey data 3D graphics are sometimes used to provide multi angle analysis of statistical data.

C) Computer art:

CG plays a major role in fine art and commerical art various computer methods. Symbolic package (AD packages are used by designer to generate computer art. Using such packages the artist can easily design shapes and can add motion.

D) Entertainment:

CG are commonly used in film industries for making animation and cartoon movies, for providing effects to the scene, television, etc. Sometimes the character itself is displayed as graphic object and sometimes it is combined with real scene and synthesized objects.

E) Education and training:

Another application of CG is education and training Nowadays in market many educational toolkits are available for child education. they can learn through videos and animation. Training a person for vechile such as driving using such simulators saves his time and efforts.

(iii) Differentiate between random scan display and raster scan display

Raster Scan System	Random Scan System
1. The electron beam scans the entire screen to draw a picture.	The electron beam scans only the part of the screen where pic information is present.
2. The video controller is required.	Video controller is not required.
3. Used to display a dynamic scene.	Used to display static picture.
4. Scan conversion is required.	Scan conversion is not required.
5. The refresh rate is independent of no. of objects in scene.	When numbers of primitives are too large random scan device flickers.
6. Pixel / spatial location of screen is used to draw an image.	Mathematical functions are used to draw an image.
7. Economical	Costly.

Q) What is Aliasing? Explain techniques of antialiasing.

⇒ When straight lines are drawn on monitor it appears zig-zag. This effect is called aliasing, happenens due to poor algorithm or hardware limitation.
zig-zag lines on screen gives the illusion of smooth line that effect is called anti-aliasing. Line or on paper looks smooth and continuous by when they are rendered on monitor screen. Creates discrete zig-zag appearance due to grid structure.
We can apply certain technique which will reduce the aliasing effect. We can achieve antialiasing either by increasing the screen resolution or by applying smart algorithm to complete the pixel locations on the shape boundary.

Techniques of antialiasing:

① Pefiltering: It is known as area sampling. In this, intensity of pixel is determined by area of pixel covered by the object.

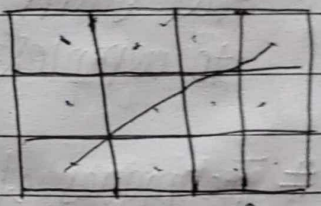
a) Unweighted area sampling:
A line passes through two pixels and algorithm illuminates the pixel which is closer to the line unweighted line sampling is extension of this concept which reduces the aliasing effect. This method illuminates both the pixel with diff. intensities.

2003085 | C21
Harsh Kasliwal.

Page No.
Date

b) Weighted area sampling: (Super sampling)

In this method, single pixel is mapped to imagine high resolution mask. Each mask position is weight as per the pixel position covered by the mask.
Pixel is divided into imaginary high resolution mask. Each mask location is weight as per the the pixel position covered by the mask.
Pixel is divided into imaginary grid and each grid location is assigned weight according to area covered by the pixel.
Average intensity of imaginary grid is placed back to original pixel location.



Original line on the screen is shown. It is very much clear that the line does not interpolate pixels perfectly. Some of the pixels are on the edge of the line, some are on the line and some are missing the line.

line with original pixel position.

1/3	2/3	1
3/3	1	0
2/3	5/7	0

Whereas the supersampled and enhanced views of some random pixel, we have divided the pixel in 3x3 imaginary grid. Value in each cell indicates the percentage of area covered by line pixel.

view of super sampled pixel.

Harsh Kesliwal
2009035
(2)

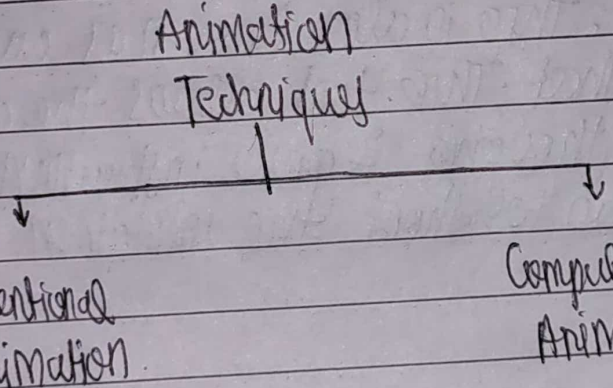
CO1

Page No.
Date

Written Assignment 2

Q.1. The animation is widely used in movies advertisement games etc. Animation gives life to the objects present in the scene through various transformation operations animation can be computer assisted or computer generated.

The idea behind the animation is to fool the human eye by displaying images at a certain speed so that they are interpreted as video of moving objects.
Animation techniques can be classified as conventional animation and computer based animation.



→ In conventional, all the frames of video are assigned and designed by hand. Frames are displayed at the rate of atleast 24 frames per second the conventional method takes a tremendous amount of time & effort to create a video.

→ Steps:

Storyboard: The sequence of animation is first drafted on paper in the form of sketches which is called a storyboard.

→ Keyframes : keyframes are the first frame of every shot in the sequence. Keyframes are used to interpolate the animation sequence. As the animation is achieved using keyframes, such type of animation is also known as keyframe animation.

→ Inbetween : Inbetween are the set of frames b/w successive keyframes. Objects in motion gradually move in subsequent frames and reach to the destination in keyframes.

→ Cels : Instead of applying animation to all the objects together, the animation scene is decomposed into small plots called cels. These cels may be independent of partially dependent.

→ Exposure sheet : This is also known as camera instruction sheet or x-sheet. This tool allows the animator to organize his thoughts & give instruction to the camera operator how to shoot the animation.

Page No.
Date

(2) Parallel Projection	Perspective Proj.
1] Projectors are parallel to each other	Projectors are not parallel
2] Need to specify the direction of projection	Centre of projection is at a finite distance need to be specify
3] Does not produce a realistic view.	Produces realistic view.
4] Centre of projection is at infinite dist.	Centre of projection is at finite distance
5] Depth information is lost.	Depth information is preserved.
6] Preserve relative proportion of objects.	Does not preserve relative proper. of object.
7] Subtypes: Orthographic projection, oblique projection.	Subtypes: single point, two point three point projection.

(3) In the Bezier curve the degree of polynomial is always one less than the number of control points. So we must have more than four points for cubic Bezier curves.

B-spline curves are the most widely used class of curves for approximating the shape due to its following properties:

→ Degree of a polynomial is independent of a no. of control points.
→ They have local control over the curves & surfaces.
→ However, derivation and generation of B-spline are more complex
→ Blending function - (n+1) control pts.

$$P(t) = \sum_{i=0}^{n} P_i \cdot B_{i,d}(t), \quad t_{min} \le t \le t_{max} \text{ and } 2 \le d \le n+1$$

→ B-spline blending f:n $B_{i,d}$ are polynomials with degree d-1. Degree d can take any integer value of control points.

→ Blending f:n for B-spline curve are defined using Cox-deboor recursive formula as shown below:

$$B_{i,1}(t) = \begin{cases} 1 & \text{if } t_i \le t \le t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$B_{i,d}(t) = \frac{t - t_i}{t_{i+d-1} - t_i} (t) + \frac{t_{i+d} - t}{t_{i+d} - t_{i+1}} (B_{i+1,d-1}(t))$$

Page No.
Date

④ physical description of the object goes through multiple sequences as shown

→ Window to the viewport or to transformation is necessary because the size of the window and viewport may not be the same all the time. So actual scene selected by window needs to be resized to fit it in the viewport.

→ Let (xw_{min}, yw_{min}) and (xw_{max}, yw_{max}) represent the lower left & upper top corner points of clipping window resp.

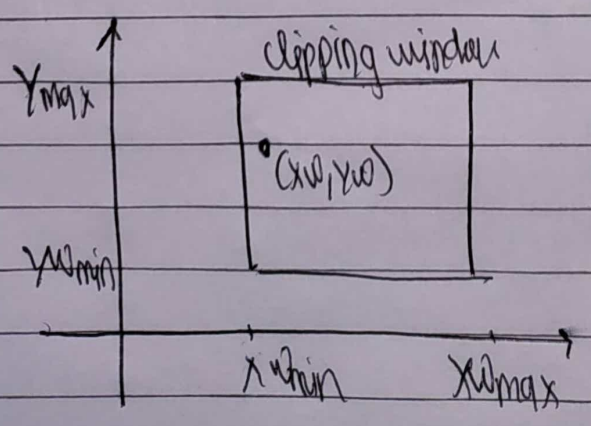
→ And let (xv_{min}, yv_{min}) and (xv_{max}, yv_{max}) represent the lower left and upper top corner points of viewport respectively.

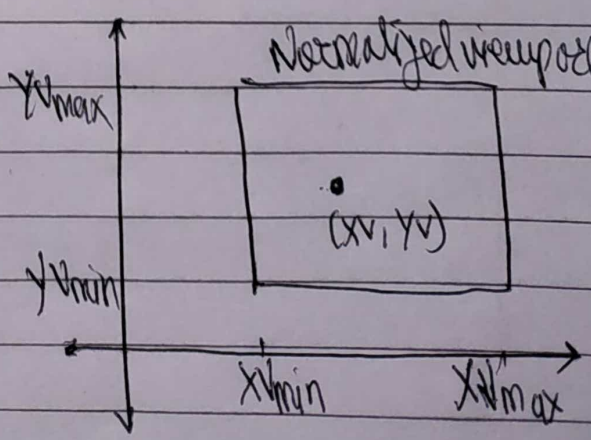
→ To maintain the same relative placement in the viewport as in a window, we normalize both.

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$xv - xv_{min} = (xv_{max} - xv_{min}) \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$xv = xv_{min} + (x_w - xv_{min})\, s_x.$$





Harsh
20030085

Similarly

$$y_v = yv_{min} + (yw - yw_{min}) \cdot S_y$$

where

$$S_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$

$$S_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$