



**AMITY UNIVERSITY**  
UTTAR PRADESH

## **COMPUTER GRAPHICS [CSE203] Practical Lab File**

**Submitted by:-**

**Name: Anchal kumari**

**Enrollment no./Sec: A12405218083(6cse3x)**

**Submitted To:-**

**Dr. Renuka Nagpal**



**AMITY SCHOOL OF ENGINEERING AND  
TECHNOLOGY AMITY UNIVERSITY CAMPUS,  
SECTOR-125, NOIDA-201301**

## INDEX

SNO	PROGRAM NAME	PERFORM DATE	SUBMISSION DATE	FACULTY SIGNATURE
1	a). Write a program to draw house using graphics. b). Write a program to make bouncing ball using graphics.	21-12-2020	12-01-2021	
2	Write a program to implement DDA line drawing algorithm.	04-01-2021	12-01-2021	
3	a).Write a program to implement Bresenham's line drawing algorithm. (for $m < 1$ ) b).Write a program to implement Bresenham's line drawing algorithm. (for $m > 1$ )	11-01-2021	12-01-2021	
4	Write a program to implement Bresenham's circle drawing algorithm.	18-01-2021	18-01-2021	
5	Write a program to implement midpoint circle drawing algorithm.	18-01-2021	18-01-2021	
6	Write a program to implement midpoint ellipse drawing algorithm.	25-01-2021	25-01-2021	
7	Write a program to implement 1. Translation of the triangle. 2. Rotation of the triangle. 3. Scaling of the triangle.	01-02-2021	01-02-2021	
8	a. Write a program to implement	08-02-2021	08-02-2021	

	Reflection of the triangle b. Reflection about arbitrary line			
9	Implement shearing of an object.	15-02-2021	15-02-2021	
10	Write a program to implement window to viewport transformation	22-02-2021	22-02-2021	
11	Write a program to implement cohen Sutherland line clipping algorithm	01-03-2021	01-03-2021	
12	Write a program to implement Sutherland - hodgeman polygon clipping algorithm	08-03-2021	08-03-2021	
13	EXTRA WORK a. Implement hyperbola b. Implement Bezier curve	15-02-2021	15-02-2021	
14	Open ended experiment	15-03-2021	22-03-21	

## **PROGRAM -1(a)**

**PROBLEM STATEMENT:** Write a program to draw house using graphics.

### **THEORY:**

**Approach:** We will create a house with the help of several lines and rectangles. Below are the steps:

1. We will draw a line in graphics by passing 4 numbers to line() function as:  
line(a, b, c, d)  
The above function will draw a line from coordinates (a, b) to (c, d) in the output window.
2. I will draw a rectangle in graphics by passing 4 numbers to rectangle() function as:  
line(left, top, right, bottom)  
The above function will draw a rectangle with coordinates of left, right, top and bottom.
3. The setfillstyle()function which sets any fill pattern in any shape created in C program using graphics.
4. The floodfill()function is used to fill an enclosed area with any color.

### **CODE:**

```
#include <conio.h>

#include <graphics.h>

#include <stdio.h>


// Driver Code

int main()
{
    // Initialize of gdriver with
    // DETECT macros
    int gdriver = DETECT, gmode;

    // Initialize structure of
    // the house
    initgraph(&gdriver, &gmode, "");
```

```
// Create lines for structure
// of the House
line(100, 100, 150, 50);

line(150, 50, 200, 100);

line(150, 50, 350, 50);
line(350, 50, 400, 100);

// Draw rectangle to give proper
// shape to the house
rectangle(100, 100, 200, 200);
rectangle(200, 100, 400, 200);
rectangle(130, 130, 170, 200);
rectangle(250, 120, 350, 180);

// Set color using setfillstyle()
// which take style and color as
// an argument
setfillstyle(2, 3);

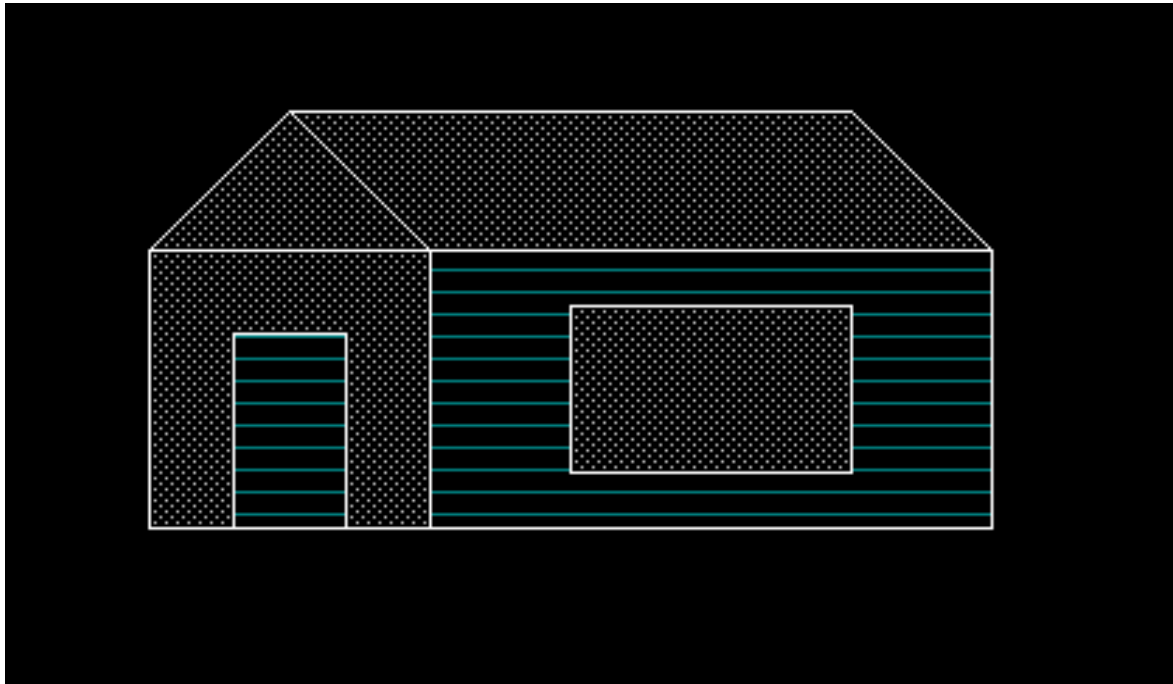
// Fill the shapes with colors white
floodfill(131, 131, WHITE);
floodfill(201, 101, WHITE);

// Change the filling color
setfillstyle(11, 7);

// Fill the shapes with changed colors
floodfill(101, 101, WHITE);
```

```
floodfill(150, 52, WHITE);  
floodfill(163, 55, WHITE);  
floodfill(251, 121, WHITE);  
  
// Close the initialized gdriver  
getch();  
closegraph();  
  
}
```

**OUTPUT:**



## **PROGRAM -1(b)**

**PROBLEM STATEMENT:** Write a program to make bouncing ball using graphics.

### **CODE:**

```
include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <dos.h>


int main()
{
    int gd = DETECT, gm;
    int i, x, y, flag=0;
    initgraph(&gd, &gm, "C:\\TC\\BGI");

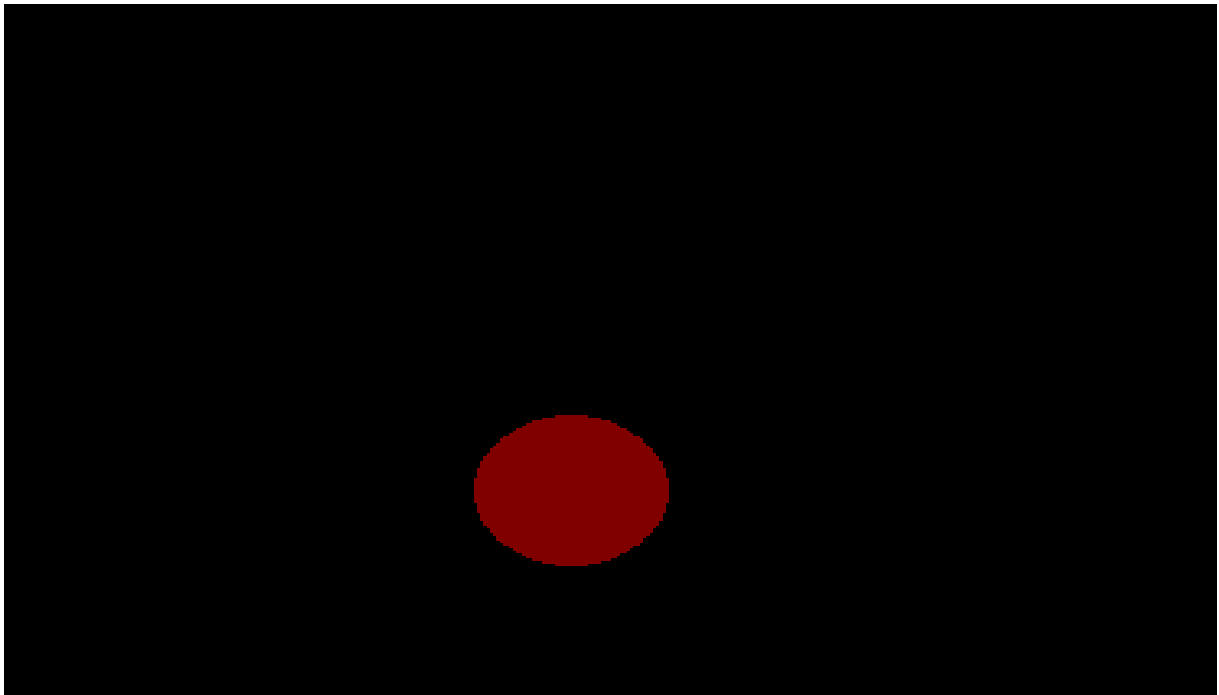
    x = getmaxx()/2;
    y = 30;
    while (!kbhit())
    {
        if(y >= getmaxy()-30 || y <= 30)
            flag = !flag;
        /* draws the gray board */
        setcolor(RED);
        setfillstyle(SOLID_FILL, RED);
        circle(x, y, 30);
        floodfill(x, y, RED);

        delay(50);
        cleardevice();
        if(flag)
```

```
{  
    y = y + 5;  
}  
else  
{  
    y = y - 5;  
}  
}
```

```
    getch();  
    closegraph();  
    return 0;  
}
```

**OUTPUT:**





## **PROGRAM -2**

**PROBLEM STATEMENT:** Write a program to implement DDA line drawing algorithm.

**THEORY:** DDA stands for Digital Differential Analyzer. It is an incremental method of scan conversion of line. In this method calculation is performed at each step but by using results of previous steps.

**Case1:** When  $|M| < 1$  then (assume that  $x_1 < x_2$ )

$x = x_1, y = y_1$  set  $\Delta x = 1$

$y_{i+1} = y_1 + m, \quad x = x + 1$

Until  $x = x_2$

**Case2:** When  $|M| > 1$  then (assume that  $y_1 < y_2$ )

$x = x_1, y = y_1$  set  $\Delta y = 1$

$x_{i+1} = x_1 + 1/m, \quad y = y + 1$

Until  $y \rightarrow y_2$

### **Advantage:**

1. It is a faster method than method of using direct use of line equation.
2. This method does not use multiplication theorem.
3. It allows us to detect the change in the value of x and y ,so plotting of same point twice is not possible.
4. This method gives overflow indication when a point is repositioned.
5. It is an easy method because each step involves just two additions.

### **Disadvantage:**

1. It involves floating point additions rounding off is done. Accumulations of round off error cause accumulation of error.
2. Rounding off operations and floating point operations consumes a lot of time.
3. It is more suitable for generating line using the software. But it is less suited for hardware implementation.

## **CODE:**

```
#include<graphics.h>

#include<conio.h>

#include<stdio.h>

int main()
{
    int gd = DETECT ,gm, i;

    float x, y,dx,dy,steps;

    int x0, x1, y0, y1;

    initgraph(&gd, &gm, "C:\\TC\\BGI");

    setbkcolor(WHITE);

    x0 = 100 , y0 = 200, x1 = 500, y1 = 300;

    dx = (float)(x1 - x0);

    dy = (float)(y1 - y0);

    if(dx>=dy)
    {
        steps = dx;
    }
    else
    {
        steps = dy;
    }

    dx = dx/steps;

    dy = dy/steps;

    x = x0;

    y = y0;

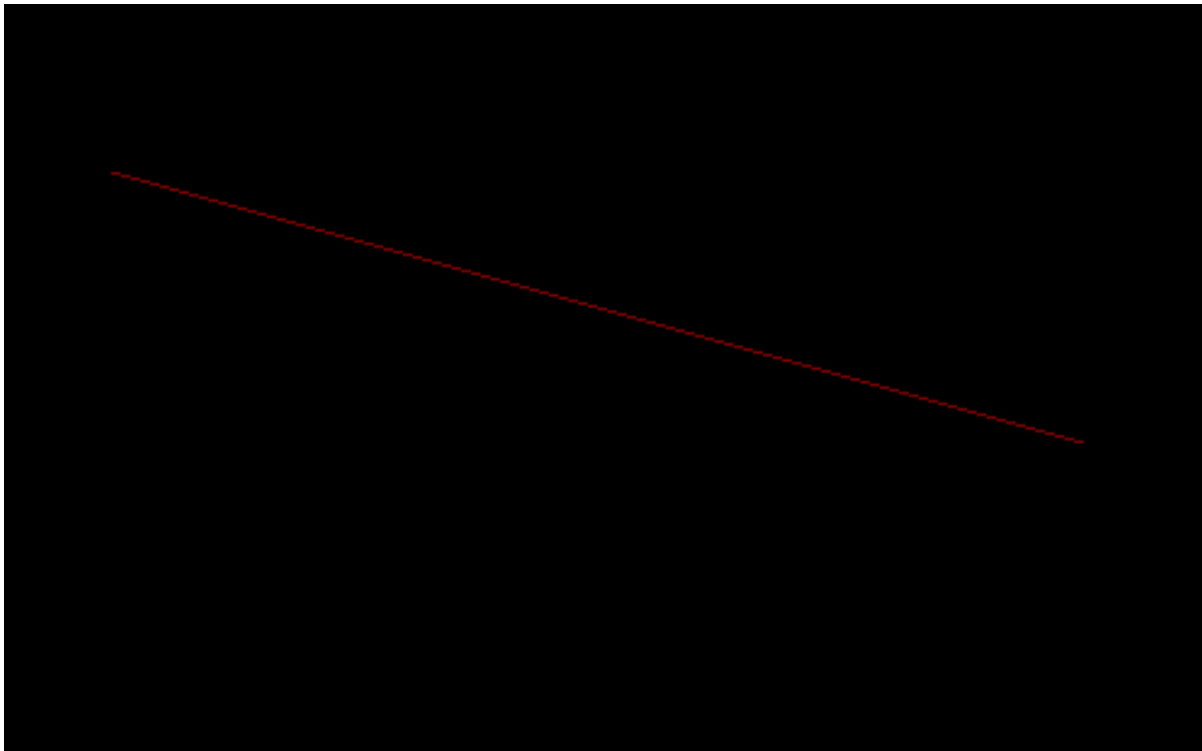
    i = 1;

    while(i<= steps)
    {

        putpixel(x, y, RED);
```

```
        x += dx;  
        y += dy;  
        i=i+1;  
    }  
    getch();  
    closegraph();  
}
```

### **OUTPUT:**



### **PROGRAM -3(a)**

**PROBLEM STATEMENT:** Write a program to implement Bresenham's line drawing algorithm. (for  $m < 1$ )

#### **THEORY:**

This algorithm is used for scan converting a line. It was developed by Bresenham. It is an efficient method because it involves only integer addition, subtractions, and multiplication operations. These operations can be performed very rapidly so lines can be generated quickly.

In this method, next pixel selected is that one who has the least distance from true line.

#### **Bresenham's Line Algorithm:**

**Step1:** Start Algorithm

**Step2:** Declare variable  $x_1, x_2, y_1, y_2, d, i_1, i_2, dx, dy$

**Step3:** Enter value of  $x_1, y_1, x_2, y_2$   
Where  $x_1, y_1$  are coordinates of starting point  
And  $x_2, y_2$  are coordinates of Ending point

**Step4:** Calculate  $dx = x_2 - x_1$   
Calculate  $dy = y_2 - y_1$   
Calculate  $i_1 = 2 * dy$   
Calculate  $i_2 = 2 * (dy - dx)$   
Calculate  $d = i_1 - dx$

**Step5:** Consider  $(x, y)$  as starting point and  $x_{end}$  as maximum possible value of  $x$ .  
If  $dx < 0$   
Then  $x = x_2$   
 $y = y_2$   
 $x_{end} = x_1$   
If  $dx > 0$   
Then  $x = x_1$   
 $y = y_1$   
 $x_{end} = x_2$

**Step6:** Generate point at  $(x, y)$  coordinates.

**Step7:** Check if whole line is generated.  
If  $x \geq x_{end}$   
Stop.

**Step8:** Calculate co-ordinates of the next pixel  
If  $d < 0$   
Then  $d = d + i_1$   
If  $d \geq 0$

Then  $d = d + i_2$   
Increment  $y = y + 1$

**Step9:** Increment  $x = x + 1$

**Step10:** Draw a point of latest (x, y) coordinates

**Step11:** Go to step 7

**Step12:** End of Algorithm

### **Advantage:**

1. It involves only integer arithmetic, so it is simple.
2. It avoids the generation of duplicate points.
3. It can be implemented using hardware because it does not use multiplication and division.
4. It is faster as compared to DDA (Digital Differential Analyzer) because it does not involve floating point calculations like DDA Algorithm.

### **Disadvantage:**

1. This algorithm is meant for basic line drawing only Initializing is not a part of Bresenham's line algorithm. So to draw smooth lines, you should want to look into a different algorithm.

### **CODE:**

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

int main()
{
    int gd=DETECT,gm;
    int i,xmid,ymid,x1,y1,x2,y2,x,y,dy,dx,p,gap=50,temp;
    float m;
    char str[3];
    initgraph(&gd,&gm,"..\\bgi");
```

```
printf("Enter first co-ords of line\n");
scanf("%d",&x1);
scanf("%d",&y1);
printf("Enter second co-ords of line\n");
scanf("%d",&x2);
scanf("%d",&y2);
if (y1>y2) // needed for -ve slope to work
{
temp=x1;
x1=x2;
x2=temp;
temp=y1;
y1=y2;
y2=temp;
}
dy=abs(y2-y1);
dx=abs(x2-x1);
m=(float)(dy)/(dx);
dy=abs(dy);
dx=abs(dx);
xmid= getmaxx()/2;
ymid =getmaxy()/2;
x2=x2+xmid;
y2=ymid-y2;
x=x1;y=y1;
if(m>1.0)
{
x=y1;
y=x1;
}
```

```
line(5,ymid,getmaxx()-5,ymid);
line(xmid+3,5,xmid+3,getmaxy()-5);

for( i= xmid+gap;i<getmaxx()-5;i=i+gap)
{
    outtextxy(i,ymid-3,"|");
    itoa(i-xmid,str,10);
    outtextxy(i,ymid+3,str);
}
for( i= ymid-gap;i>5;i=i-gap)
{
    outtextxy(xmid,i,"-");
    itoa(ymid-i,str,10);
    outtextxy(xmid+5,i,str);

}
for( i= xmid-gap;i>5;i=i-gap)
{
    outtextxy(i,ymid-3,"|");
    itoa(-(xmid-i),str,10);
    outtextxy(i-6,ymid+3,str);

}
for( i= ymid+gap;i<getmaxy()-5;i=i+gap)
{
    outtextxy(xmid,i,"-");
    itoa(-(i-ymid),str,10);
    outtextxy(xmid+8,i,str);
}
```

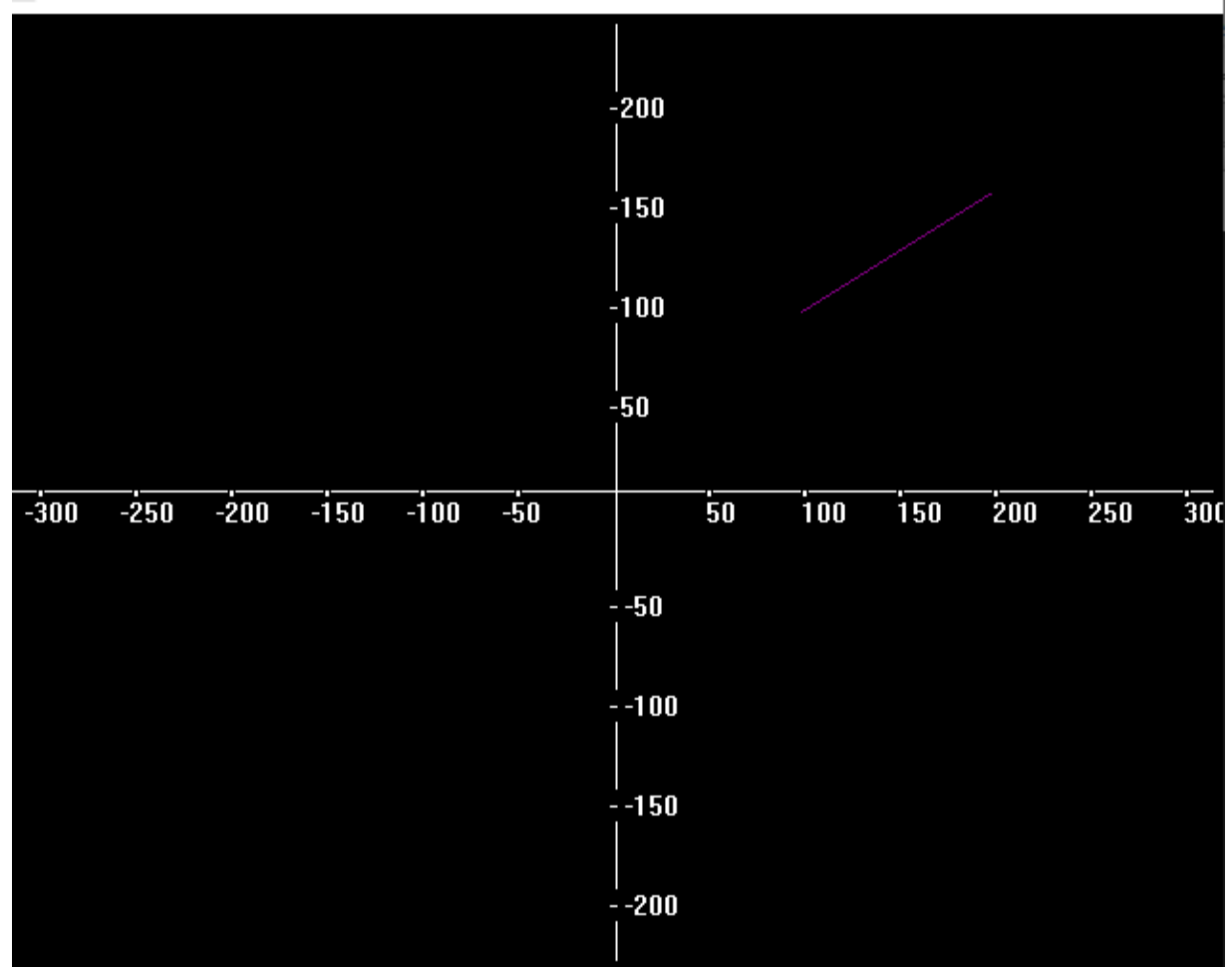
```
x=x+xmidx;
y=ymid -y;
p=2*dy-dx;
while(x<x2)
{
if(p>=0)
{
putpixel(x,y,5);

y=y-1;
p=p+2*dy-2*dx;
}
else
{
putpixel(x,y,5);
p=p+2*dy;
}
x=x+1;
}
getch();
closegraph();
}
```



**OUTPUT:**

```
Enter first co-ords of line
100
90
Enter second co-ords of line
200
150
```



### **PROGRAM -3(b)**

**PROBLEM STATEMENT:** Write a program to implement Bresenham's line drawing algorithm. (for  $m > 1$ )

#### **CODE:**

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

int main()
{
    int gd=DETECT,gm;
    int i,xmid,ymid,x1,y1,x2,y2,x,y,dy,dx,p,gap=50,temp;
    float m;
    char str[3];
    initgraph(&gd,&gm,"..\\bgi");

    printf("Enter first co-ords of line\n");
    scanf("%d",&x1);
    scanf("%d",&y1);
    printf("Enter second co-ords of line\n");
    scanf("%d",&x2);
    scanf("%d",&y2);
    if (y1>y2) // needed for -ve slope to work
    { temp=x1;
      x1=x2;
      x2=temp;
      temp=y1;
      y1=y2;
      y2=temp;
    }
    dy=abs(y2-y1);
```

```

dx=abs(x2-x1);
m=(float)(dy)/(dx);
dy=abs(dy);
dx=abs(dx);
xmid= getmaxx()/2;
ymid =getmaxy()/2;
x2=x2+xmid;
y2=ymid-y2;
x=x1;y=y1;
if(m>1.0)
{
x=y1;
y=x1;
}

line(5,ymid,getmaxx()-5,ymid);
line(xmid+3,5,xmid+3,getmaxy()-5);

for( i= xmid+gap;i<getmaxx()-5;i=i+gap)
{
outtextxy(i,ymid-3,"|");
itoa(i-xmid,str,10);
outtextxy(i,ymid+3,str);
}
for( i= ymid-gap;i>5;i=i-gap)
{
outtextxy(xmid,i,"-");
itoa(ymid-i,str,10);
outtextxy(xmid+5,i,str);
}

```

```

    }
    for( i= xmid-gap;i>5;i=i-gap)
    {
        outtextxy(i,ymid-3,"|");
        itoa(-(xmid-i),str,10);
        outtextxy(i-6,ymid+3,str);

    }
    for( i= ymid+gap;i<getmaxy()-5;i=i+gap)
    {
        outtextxy(xmid,i,"-");
        itoa(-(i-ymid),str,10);
        outtextxy(xmid+8,i,str);
    }

    x=x+xmid;
    y=ymid -y;
    p=2*dx-dy;
    while(x<x2)
    {
        if(p>=0)
        {
            putpixel(x,y,5);

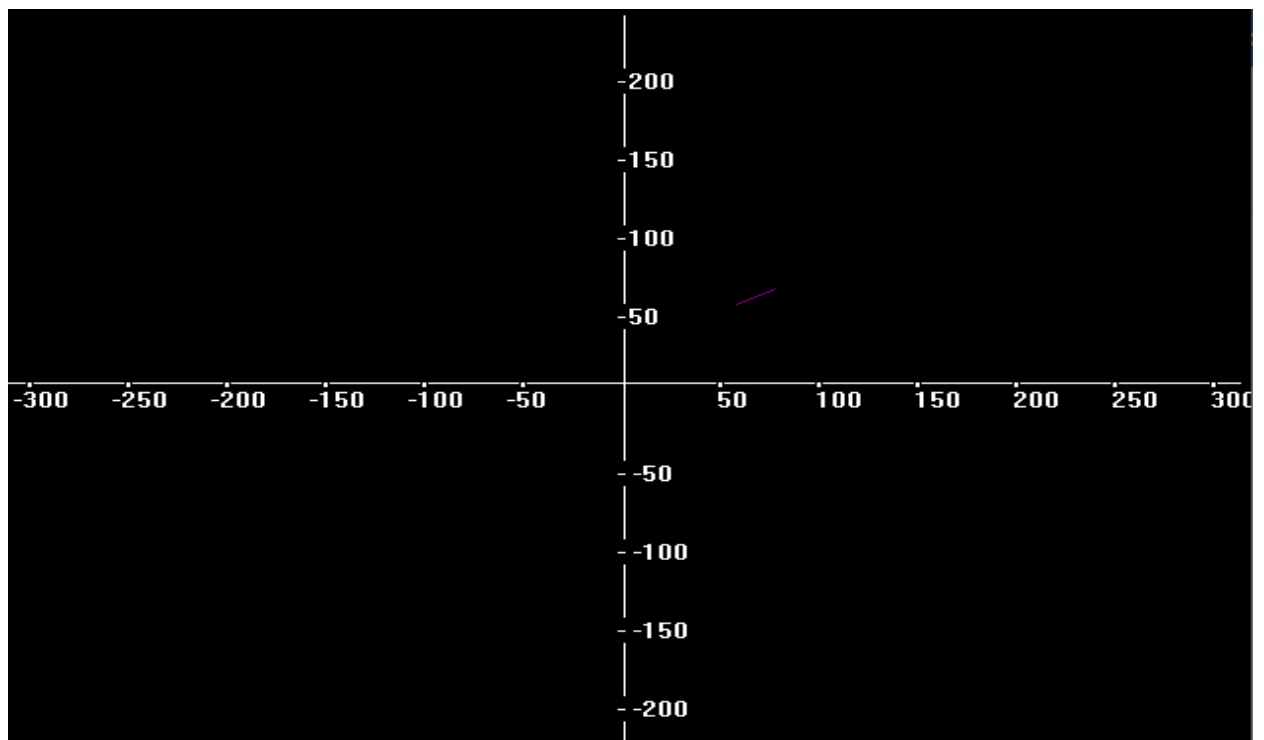
            y=y-1;
            p=p+2*dx-2*dy;
        }
        else
        {

```

```
putpixel(x,y,5);  
p=p+2*dx;  
}  
x=x+1;  
}  
getch();  
closegraph();  
}
```

### **OUTPUT:**

```
Enter first co-ords of line  
50  
60  
Enter second co-ords of line  
80  
120
```



## **PROGRAM -4**

**PROBLEM STATEMENT:** Write a program to implement Bresenham's circle drawing algorithm.

### **THEORY:**

Scan-Converting a circle using Bresenham's algorithm works as follows: Points are generated from  $90^\circ$  to  $45^\circ$ , moves will be made only in the +x & -y directions

The best approximation of the true circle will be described by those pixels in the raster that falls the least distance from the true circle. We want to generate the points from  $90^\circ$  to  $45^\circ$ . Assume that the last scan-converted pixel is  $P_1$  as shown in fig. Each new point closest to the true circle can be found by taking either of two actions.

1. Move in the x-direction one unit or
2. Move in the x- direction one unit & move in the negative y-direction one unit.

### **Bresenham's Circle Algorithm:**

**Step1:** Start Algorithm

**Step2:** Declare p, q, x, y, r, d variables

p, q are coordinates of the center of the circle  
r is the radius of the circle

**Step3:** Enter the value of r

**Step4:** Calculate  $d = 3 - 2r$

**Step5:** Initialize      $x=0$   
                            &nbsp; $y=r$

**Step6:** Check if the whole circle is scan converted  
      If  $x \geq y$   
      Stop

**Step7:** Plot eight points by using concepts of eight-way symmetry. The center is at (p, q).  
Current active pixel is (x, y).

```
putpixel (x+p, y+q)
putpixel (y+p, x+q)
putpixel (-y+p, x+q)
putpixel (-x+p, y+q)
putpixel (-x+p, -y+q)
putpixel (-y+p, -x+q)
putpixel (y+p, -x+q)
putpixel (x+p, -y-q)
```

**Step8:** Find location of next pixels to be scanned

```
If  $d < 0$ 
then  $d = d + 4x + 6$ 
increment  $x = x + 1$ 
If  $d \geq 0$ 
then  $d = d + 4(x - y) + 10$ 
increment  $x = x + 1$ 
decrement  $y = y - 1$ 
```

**Step9:** Go to step 6

**Step10:** Stop Algorithm

### **CODE:**

```
#include<graphics.h>
#include<iostream>
#include<conio.h>
using namespace std;
void Plotting(int xc,int yc,int x,int y)
{
    int xp = getmaxx()/2;
    int yp = getmaxy()/2;
    putpixel(xp+(x+xc),yp-(y+yc),RED);
    putpixel(xp+(x+xc),yp-(-y+yc),YELLOW);
    putpixel(xp+(-x+xc),yp-(-y+yc),GREEN);
    putpixel(xp+(-x+xc),yp-(y+yc),YELLOW);
    putpixel(xp+(y+xc),yp-(x+yc),12);
    putpixel(xp+(y+xc),yp-(-x+yc),14);
    putpixel(xp+(-y+xc),yp-(-x+yc),15);
    putpixel(xp+(-y+xc),yp-(x+yc),6);
}

void Brescircle(int xc,int yc,int r)
{

```

```

int x=0,y=r,d=3-(2*r);
Plotting(xc,yc,x,y);
while(x<=y)
{
    if(d<=0)
        d=d+(4*x)+6;
    else
    {
        d=d+(4*x)-(4*y)+10;
        y=y-1;
    }
    x=x+1;
    Plotting(xc,yc,x,y);
}
}
int main()
{
    int xc,yc,r;
    int gdriver = DETECT, gmode;
    initgraph(&gdriver, &gmode, (char *)"");
    cout<<"Enter the value of center :\n x : ";
    cin>>xc;
    cout<<"\n y : ";
    cin>>yc;
    cout<<"\nEnter the value of radius :";
    cin>>r;
    cleardevice();
    line(getmaxx()/2,0,getmaxx()/2,getmaxy());
    line(0,getmaxy()/2,getmaxx(),getmaxy()/2);
    Brescircle(xc,yc,r);
}

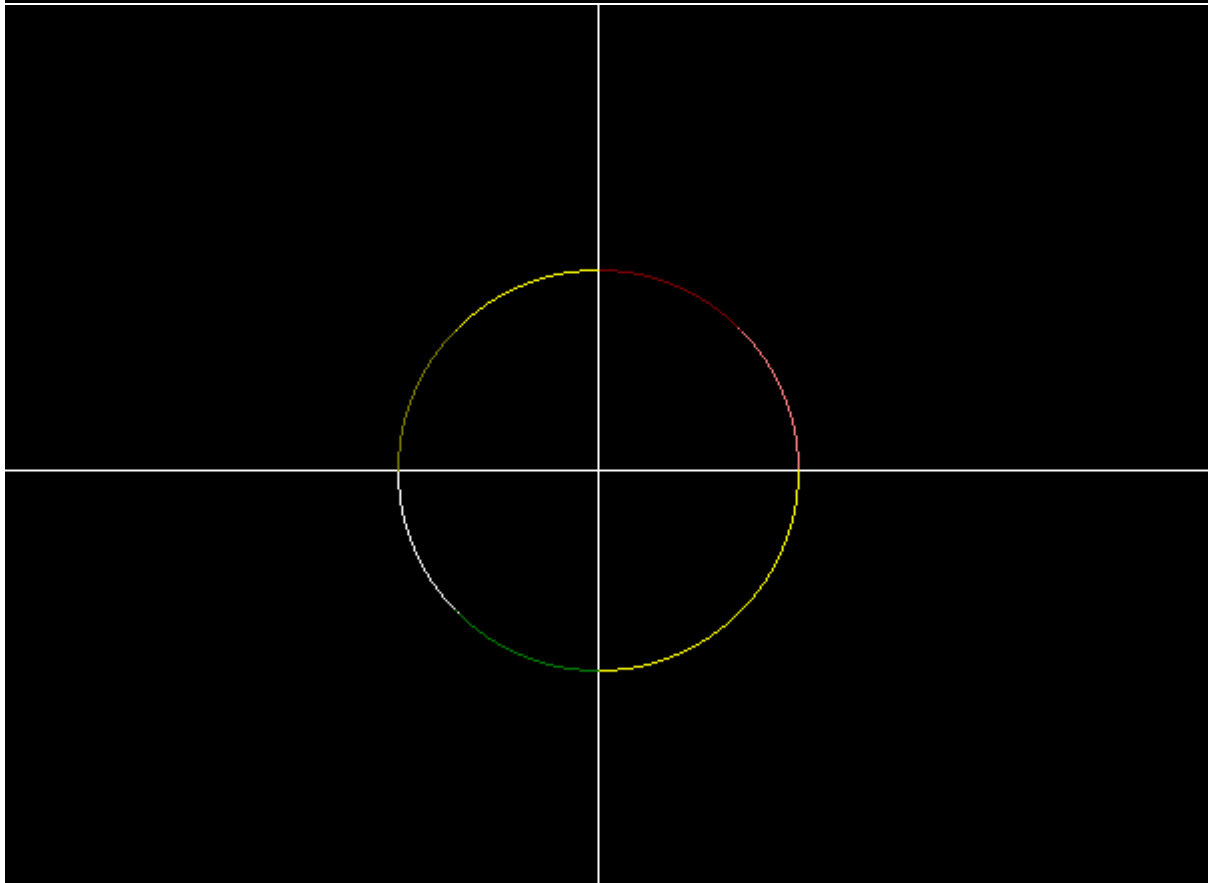
```



```
    getch();  
    return 0;  
}
```

**OUTPUT:**

```
Enter the value of center :  
x : 0  
  
y : 0  
  
Enter the value of radius :100
```



## PROGRAM -5

**PROBLEM STATEMENT:** Write a program to implement midpoint circle drawing algorithm.

### THEORY:

The **mid-point** circle drawing algorithm is an algorithm used to determine the points needed for rasterizing a circle.

We use the **mid-point** algorithm to calculate all the perimeter points of the circle in the **first octant** and then print them along with their mirror points in the other octants. This will work because a circle is symmetric about its centre.

### **Algorithm:**

**Step1:** Put  $x=0$ ,  $y=r$  in equation 2

We have  $p=1-r$

**Step2:** Repeat steps while  $x \leq y$

Plot  $(x, y)$

If  $(p < 0)$

Then set  $p = p + 2x + 3$

Else

$p = p + 2(x-y)+5$

$y = y - 1$  (end if)

$x = x + 1$  (end loop)

**Step3:** End

### CODE:

```
#include<graphics.h>
#include<iostream>
#include<conio.h>
using namespace std;
void Plotting(int xc,int yc,int x,int y)
{
    int xp = getmaxx()/2;
    int yp = getmaxy()/2;
    putpixel(xp+(x+xc),yp-(y+yc),RED);
    putpixel(xp+(x+xc),yp-(-y+yc),YELLOW);
    putpixel(xp+(-x+xc),yp-(-y+yc),GREEN);
    putpixel(xp+(-x+xc),yp-(y+yc),YELLOW);
    putpixel(xp+(y+xc),yp-(x+yc),12);
    putpixel(xp+(y+xc),yp-(-x+yc),14);
    putpixel(xp+(-y+xc),yp-(-x+yc),15);
    putpixel(xp+(-y+xc),yp-(x+yc),6);
}
```

```

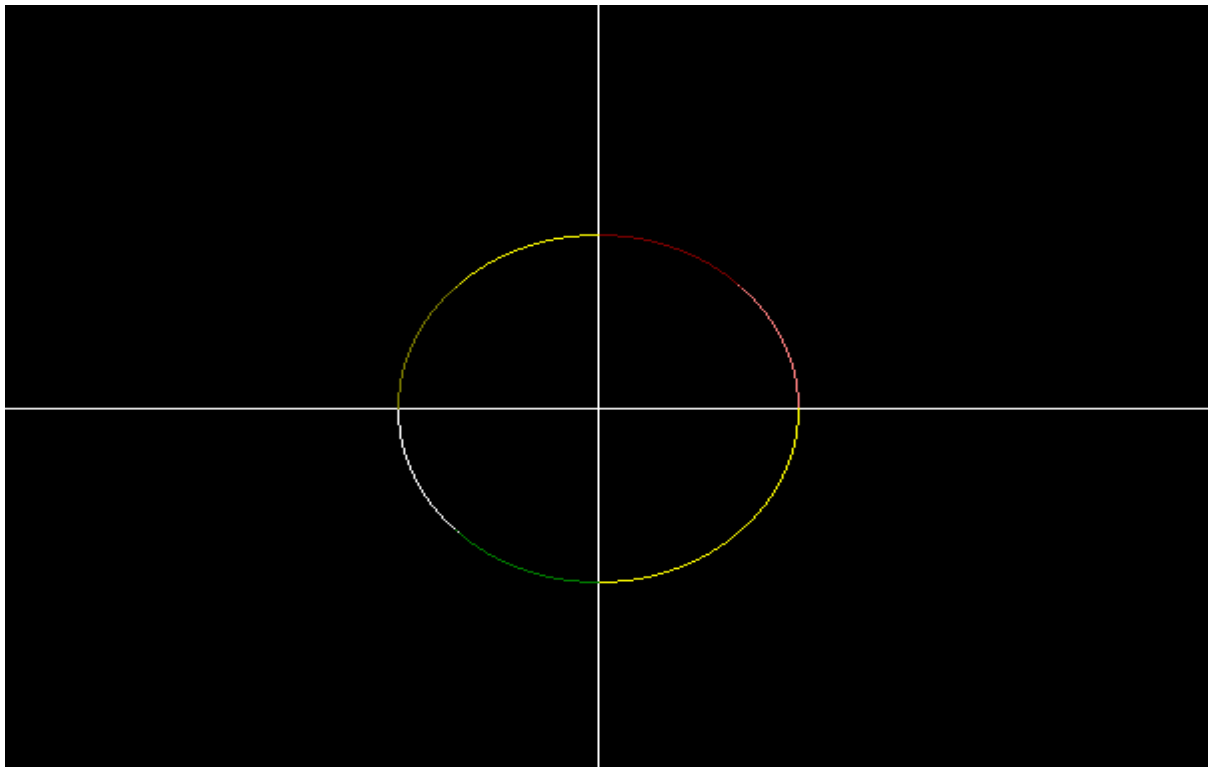
void Brescircle(int xc,int yc,int r)
{
    int x=0,y=r,d=5/4-r;
    Plotting(xc,yc,x,y);
    while(x<=y)
    {
        if(d<=0)
            d=d+(2*x)+3;
        else
        {
            d=d+(2*x)-(2*y)+5;
            y=y-1;
        }
        x=x+1;
        Plotting(xc,yc,x,y);
    }
}

int main()
{
    int xc,yc,r;
    int gdriver = DETECT, gmode;
    initgraph(&gdriver, &gmode, (char *) "");
    cout<<"Enter the value of center :\n x : ";
    cin>>xc;
    cout<<"\n y : ";
    cin>>yc;
    cout<<"\nEnter the value of radius : ";
    cin>>r;
    cleardevice();
    line(getmaxx()/2,0,getmaxx()/2,getmaxy());
    line(0,getmaxy()/2,getmaxx(),getmaxy()/2);
    Brescircle(xc,yc,r);
    getch();
    return 0;
}

```

**OUTPUT:**

```
Enter the value of center :  
x : 0  
y : 0  
Enter the value of radius :100
```



## PROGRAM -6

**PROBLEM STATEMENT:** Write a program to implement midpoint ellipse drawing algorithm.

### THEORY:

This is an incremental method for scan converting an ellipse that is centered at the origin in standard position i.e., with the major and minor axis parallel to coordinate system axis. It is very similar to the midpoint circle algorithm. Because of the four-way symmetry property we need to consider the entire elliptical curve in the first quadrant. Let's first rewrite the ellipse equation and define the function  $f$  that can be used to decide if the midpoint between two candidate pixels is inside or outside the ellipse:

$$f(x, y) = b^2x^2 + a^2y^2 - a^2b^2 = \begin{cases} < 0 (x,y) \text{ inside} \\ 0 (x,y) \text{ on} \\ > 0 (x,y) \text{ outside} \end{cases}$$

#### **Initial Decision Parameter for region1**

$$P_0 = r_y^2 (0+1)^2 + r_x^2 (r_y-1/2)^2 - r_x^2 r_y^2$$

$$P_0 = r_y^2 + r_x^2/4 - r_y r_x^2$$

#### **Initial Decision Parameter for region2**

$$P_{20} = r_y^2 (x+1/2)^2 + r_x^2 (y-1)^2 - r_x^2 r_y^2$$

### CODE:

```
#include <graphics.h>
#include<iostream>
#include<conio.h>
#include"stdlib.h"
using namespace std;
void MidpointEllipse(int a, int b, int xc, int yc)
{
    int x=0;
    int y=b;
    //Region 1
    int p=(b*b)-(a*a*b)+(0.25*a*a);
    do
    {
        putpixel(xc+x,yc+y,WHITE);
        putpixel(xc+x,yc-y,RED);
        putpixel(xc-x,yc+y,GREEN);
        putpixel(xc-x,yc-y,YELLOW);

        if(p<0)
```

```

    {
        x=x+1;
        p=p+2*b*b*x+b*b;
    }
    else
    {
        x=x+1;
        y=y-1;
        p=p+2*b*b*x-2*a*a*y+b*b;
    }
}while(2*b*b*x<2*a*a*y);
///  

//Region 2
p=(b*b*(x+0.5)(x+0.5))+((y-1)(y-1)*a*a-a*a*b*b);
do
{
    putpixel(xc+x,yc+y,WHITE);
    putpixel(xc+x,yc-y,RED);
    putpixel(xc-x,yc+y,GREEN);
    putpixel(xc-x,yc-y,YELLOW);
    if(p>0)
    {
        y=y-1;
        p=p-2*a*a*y+a*a;
    }
    else
    {
        x=x+1;
        y=y-1;
        p=p-2*a*a*y+2*b*b*x+a*a;
    }
}while(y!=0);
// */
}

int main()
{
    int h,k,a,b;
    int gd = DETECT, gm,i=0;
    initgraph(&gd, &gm,"");

    // cout<<"\n ENTER CENTER OF ELLIPSE";
    cout<<"\n ENTER CENTER  (h, k) :";
    cin>>h>>k;
    k= 240-k;
    h= 320+h;
    cout<<"\n ENTER LENGTH OF MAJOR AND MINOR AXIS";
    cin>>a>>b;

```

```
line(getmaxx()/2,0,getmaxx()/2,getmaxy());  
line(0,getmaxy()/2,getmaxx(),getmaxy()/2);
```

```
MidpointEllipse(a,b,h,k);
```

```
/*
```

```
if(y<0) y = 240-y;
```

```
else y = 240-y;
```

```
if(x>0) x = 320+x;
```

```
else x = 320+x;
```

```
*/
```

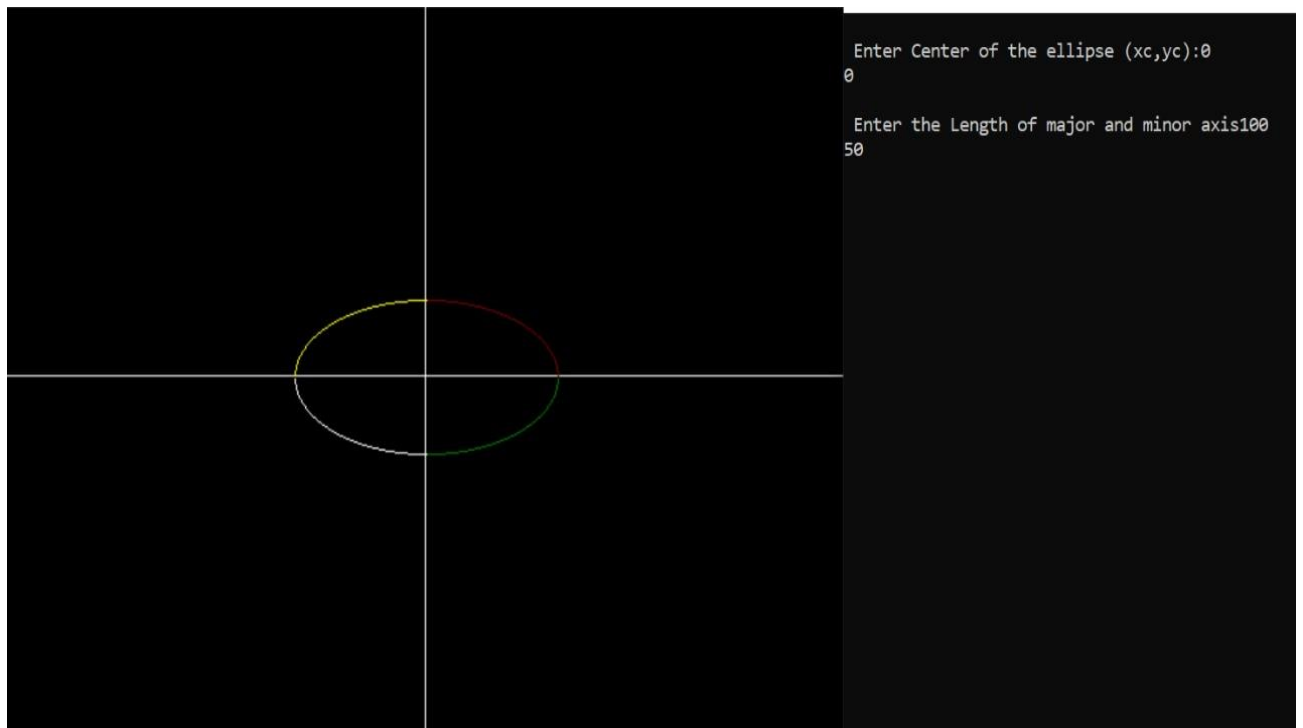
```
printf("\nOUTPUT IS DISPLAYED ON ANOTHER WINDOW:");
```

```
getch();
```

```
closegraph();
```

```
}
```

## OUTPUT:



## **PROGRAM -7**

**PROBLEM STATEMENT:** Write a program to implement Translation, Rotation, Scaling of the different shaped object

### **THEORY:**

**Translation:** It is the straight line movement of an object from one position to another is called Translation. Here the object is positioned from one coordinate location to another.

**Translation of point:** To translate a point from coordinate position (x, y) to another (x1 y1), we add algebraically the translation distances Tx and Ty to original coordinate.

$$X1=x+tx$$

$$Y1=t+ty$$

The translation pair (Tx,Ty) is called as translation vectors.

Translation is a movement of objects without deformation. Every position or point is translated by the same amount. When the straight line is translated, then it will be drawn using endpoints.

**Rotation :** It is a 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 rotation point. Rotation point is also called a pivot point. It is print about which object is rotated.

#### **Types of Rotation**

1. **Anticlockwise** - The positive value of the rotation angle rotates an object in a anti-clockwise direction.
2. **Counterclockwise** - The negative value of the pivot point (rotation angle) rotates an object in a clockwise direction.

### **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  $S_y$  in y-direction. If the original position is x and y. Scaling factors are  $S_x$  and  $S_y$  then the value of coordinates after scaling will be  $x^1$  and  $y_1$ .

If the picture to be enlarged to twice its original size then  $S_x = S_y = 2$ . If  $S_x$  and  $S_y$  are not equal then scaling will occur but it will elongate or distort the picture.

If scaling factors are less than one, then the size of the object will be reduced. If scaling factors are higher than one, then the size of the object will be enlarged.

### **CODE:**

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
int x[10],y[10],nx[10],ny[10];
```



```

double b[3][3]={ 1,0,0,
0,1,0,
0,0,1,
};
int main()
{
    int gm,gd=DETECT,n,choice,midx,midy,i,sx,sy,xt,yt,r;
    float t;
    int c[10][10];
    float a[10][10];
    initgraph(&gd,&gm,(char*)"");
    midx=getmaxx()/2;
    midy=getmaxy()/2;
    line(midx,0,midx,midy*2);
    line(0,midy,midx*2,midy);
    printf("Enter the number of sides:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        printf("\n P[%d] :",i);
        scanf("%d%d",&x[i],&y[i]);
    }
    for(i=1;i<=n;i++)
    {
        if(i==n)
            line(x[i]+midx,midy-y[i],x[1]+midx,midy-y[1]);
        else
            line(x[i]+midx,midy-y[i],x[i+1]+midx,midy-y[i+1]);
    }
    setcolor(YELLOW);

```

```

printf("\n 1.Transaction\n 2.Rotation\n 3.Scalling\n 4.Exit\n");
printf("\nEnter your choice: ");
scanf("%d",&choice);
switch(choice)
{
    case 1:
        printf("\n Enter the translation factor: ");
        scanf("%d%d",&xt,&yt);
        for(i=1;i<=n;i++)
        {
            nx[i]=x[i]+xt;
            ny[i]=y[i]+yt;
        }
        for(i=1;i<=n;i++)
        {
            if(i==n)
                line(nx[i]+midx,midy-ny[i],nx[1]+midx,midy-ny[1]);
            else
                line(nx[i]+midx,midy-ny[i],nx[i+1]+midx,midy-ny[i+1]);
        }
        break;
    case 2:
        printf("\n Enter the angle of rotation :");
        scanf("%d",&r);
        b[0][0]=cos(3.14*r/180);
        b[0][1]=sin(3.14*r/180);
        b[0][2]=0;
        b[1][0]=-sin(3.14*r/180);
        b[1][1]=cos(3.14*r/180);
        b[1][2]=0;

```

```

b[2][0]=0;
b[2][1]=0;
b[2][2]=1;
for(i=1;i<=n;i++)
{
    a[i][0]=x[i];
    a[i][1]=y[i];
    c[i][0]=a[i][0]*b[0][0]+a[i][1]*b[1][0]+b[2][0];
    c[i][1]=a[i][0]*b[0][1]+a[i][1]*b[1][1]+b[2][1];
    printf("%d%d",c[0][0],c[0][1]);
    x[i]=c[i][0];
    y[i]=c[i][1];
}
for(i=1;i<=n;i++)
{
    if(i==n)
        line(x[i]+midx,midy-y[i],x[1]+midx,midy-y[1]);
    else
        line(x[i]+midx,midy-y[i],x[i+1]+midx,midy-y[i+1]);
}
break;
case 3:
    printf("\n Enter the scaling factor: ");
    scanf("%d",&sx,&sy);
    for(i=1;i<=n;i++)
    {
        nx[i]=x[i]*sx;
        ny[i]=y[i]*sy;
    }
    for(i=1;i<=n;i++)

```

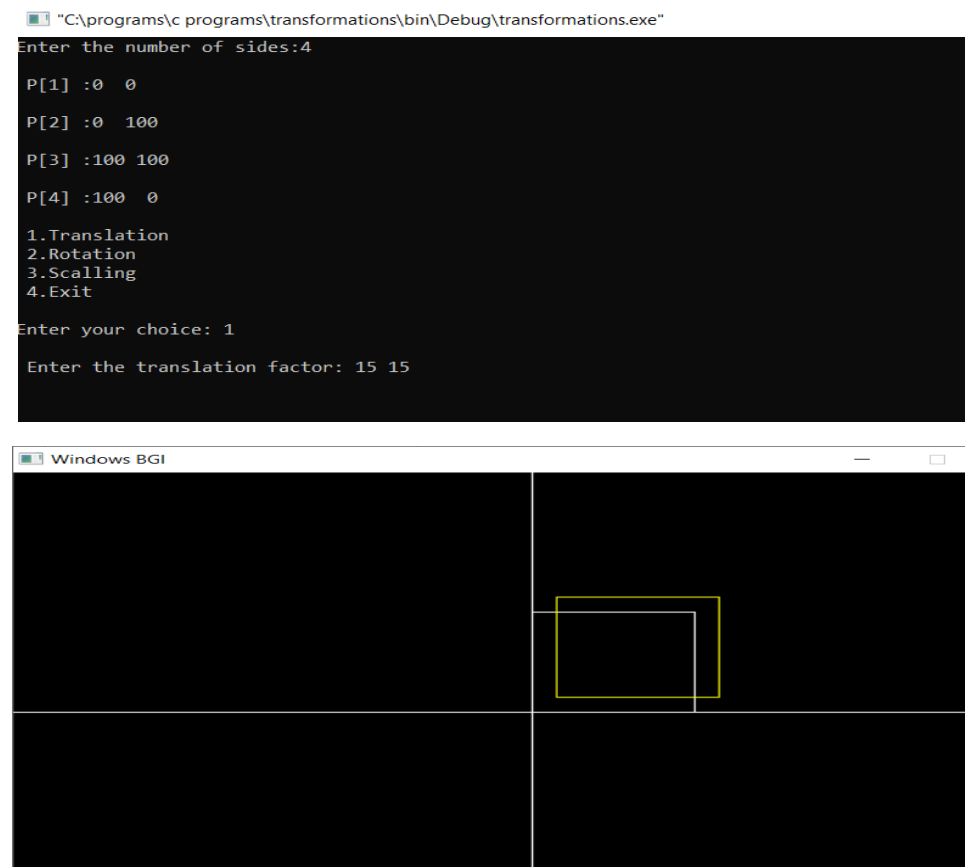
```

        {
            if(i==n)
                line(nx[i]+midx,midy-ny[i],nx[1]+midx,midy-ny[1]);
            else
                line(nx[i]+midx,midy-ny[i],nx[i+1]+midx,midy-ny[i+1]);
        }
        break;
        case 4:break;
        default: printf("\nInvalid Choice");
    }
    getch();
    closegraph();
    return 0;
}

```

## **OUTPUT:**

### **TRANSLATION:**



## ROTATION:

"C:\programs\c programs\transformations\bin\Debug\transformations.exe"

Enter the number of sides:5

P[1] :-30 -30

P[2] :30 -30

P[3] :60 70

P[4] :0 90

P[5] :-60 70

1.Translation

2.Rotation

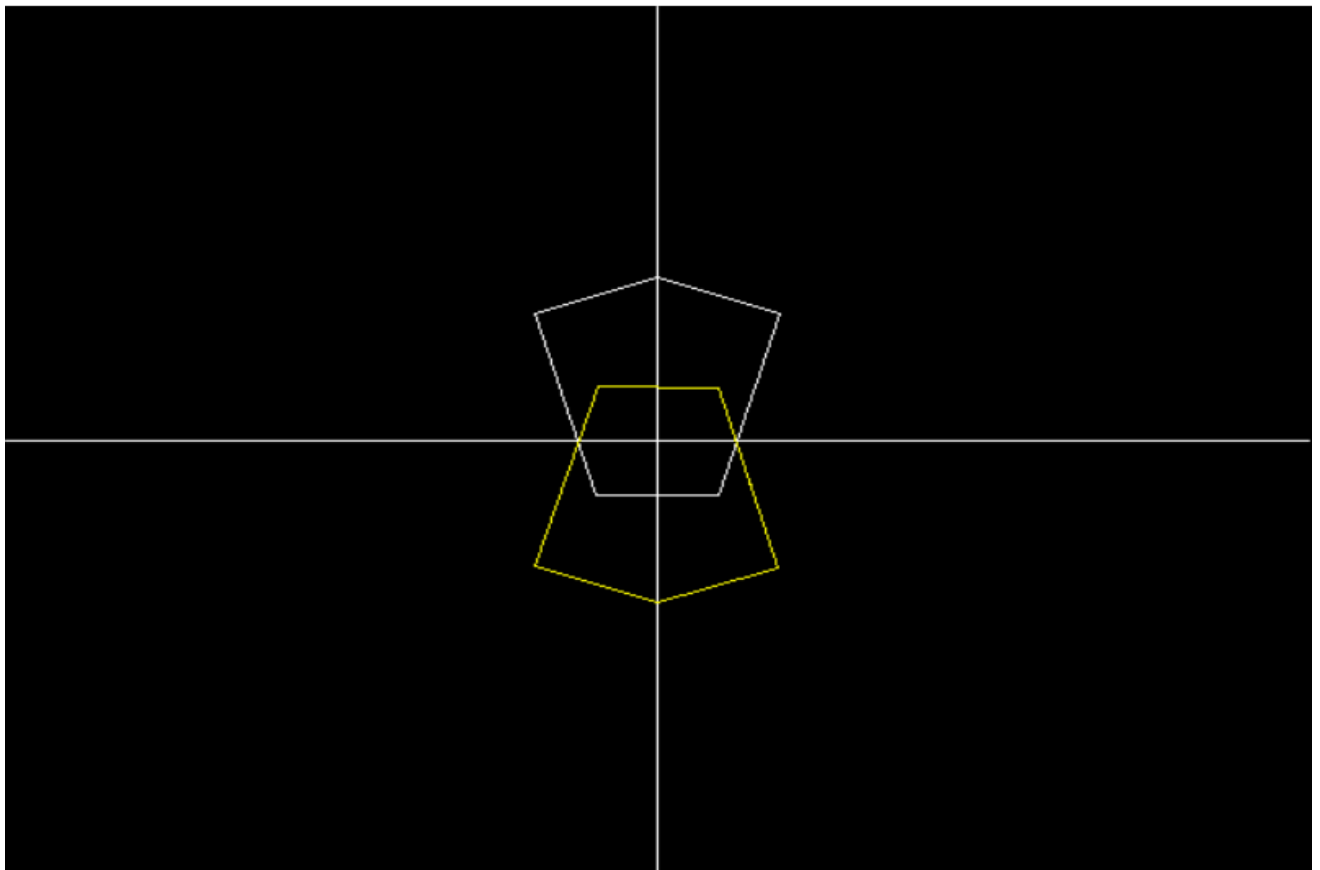
3.Scalling

4.Exit


Enter your choice: 2

Enter the angle of rotation :180

Windows BGI



## SCALING:

 "C:\programs\c programs\transformations\bin\Debug\transformations.exe"

Enter the number of sides:4

P[1] :0 40

P[2] :40 40

P[3] :40 0

P[4] :0 0

1.Translation

2.Rotation

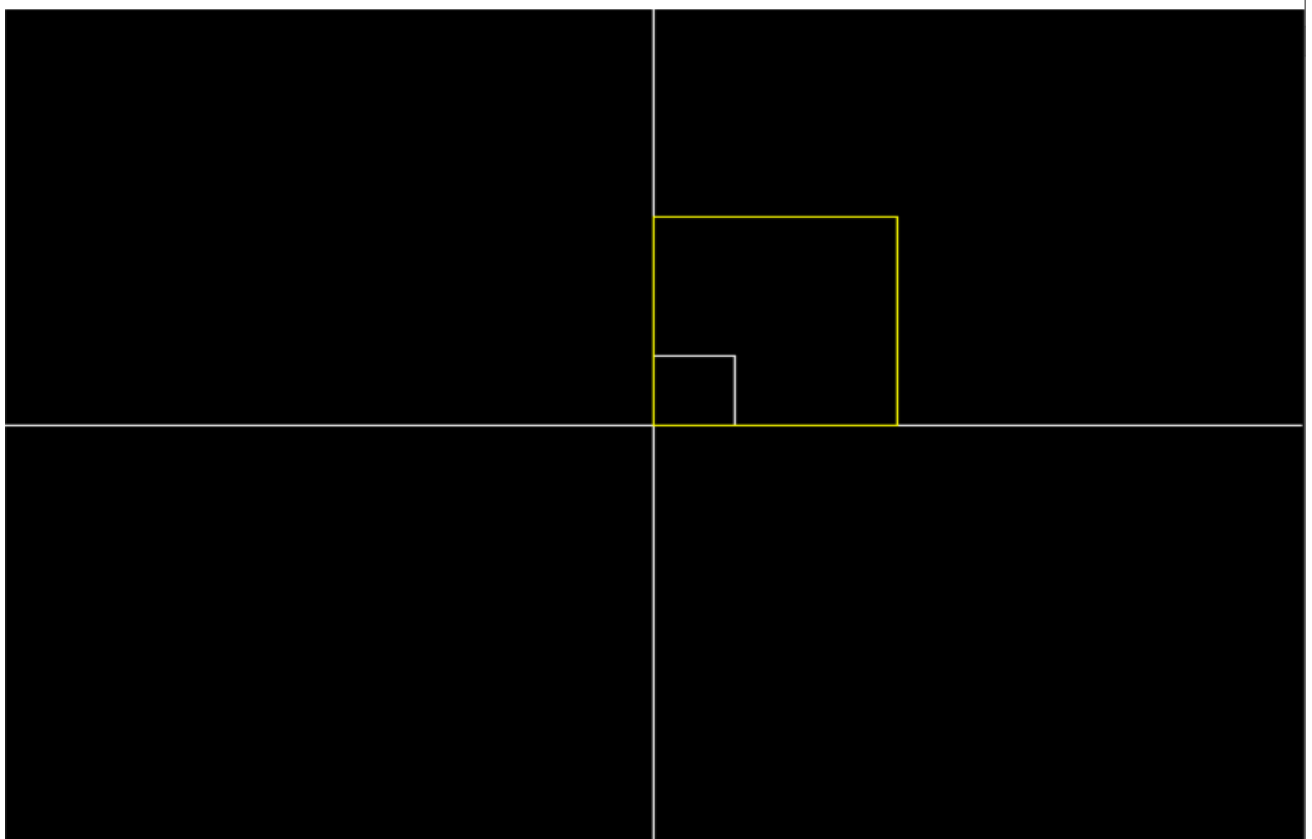
3.Scalling

4.Exit

Enter your choice: 3

Enter the scaling factor: 3 3

 Windows BGI



## **PROGRAM -8(a)**

**PROBLEM STATEMENT:** Write a program to implement Reflection of the triangle.

### **THEORY:**

It is a transformation which produces a mirror image of an object. The mirror image can be either about x-axis or y-axis. The object is rotated by 180°.

1. Reflection about the x-axis
2. Reflection about the y-axis
3. Reflection about an axis perpendicular to xy plane and passing through the origin
4. Reflection about line y=x

**Reflection about x-axis:** The object can be reflected about x-axis with the help of the following matrix

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

In this transformation value of x will remain same whereas the value of y will become negative.

**Reflection about y-axis:** The object can be reflected about y-axis with the help of following transformation matrix

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

Here the values of x will be reversed, whereas the value of y will remain the same. The object will lie another side of the y-axis.

**Reflection about an axis perpendicular to xy plane and passing through origin:**

In the matrix of this transformation is given below

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

**Reflection about line  $y=x$ :** The object may be reflected about line  $y = x$  with the help of following transformation matrix

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

**CODE:**

```
#include<iostream>

#include<graphics.h>

#include<conio.h>

#include<math.h>

using namespace std;

float mat[3][3];

float result[3][3];

int multiply_matrix(float A[3][3], float B[3][3])
{
    for(int i=0; i<3; i++)
        for(int j=0; j<3; j++)
            result[i][j] = 0;
    for(int i=0; i<3; i++)
        for(int j=0; j<3; j++)
            for(int k=0; k<3; k++)
                result[i][j] += A[i][k]*B[k][j];
}

void reflection()
{
    setcolor(WHITE);

    line(0, getmaxy()/2, getmaxx(), getmaxy()/2);

    line(getmaxx()/2, 0, getmaxx()/2, getmaxy());
```



```
setcolor(YELLOW);
```

```
line(mat[0][0] +320, 240-mat[1][0], 320+mat[0][1],240-mat[1][1]);
```

```
line(mat[0][1] +320, 240-mat[1][1], 320+mat[0][2],240-mat[1][2]);
```

```
line(mat[0][2] +320, 240-mat[1][2], 320+mat[0][0],240-mat[1][0]);
```

```
int choice;
```

```
float reflectX[3][3] = { 1, 0, 0, 0, -1, 0, 0, 0, 1};
```

```
float reflectY[3][3] = {-1, 0, 0, 0, 1, 0, 0, 0, 1};
```

```
float reflectXY[3][3] = {-1, 0, 0, 0, -1, 0, 0, 0, 1};
```

```
float reflectX_Y[3][3] = {0, 1, 0, 1, 0, 0, 0, 0, 1};
```

```
float reflect_neg[3][3] = {0, -1, 0, -1, 0, 0, 0, 0, -1};
```

```
cout<<"\nEnter no of your choice\n1. About X axis\n2. About Y axis\n3. About Origin  
XY\n4. About X=Y\n5. About X= -Y\n";
```

```
cin>>choice;
```

```
float reflect[3][3];
```

```
if(choice == 1)
```

```
    for(int i=0; i<3; i++)
```

```
        for(int j=0; j<3; j++)
```

```
            reflect[i][j] = reflectX[i][j];
```

```
else if(choice == 2)
```

```
    for(int i=0; i<3; i++)
```

```
        for(int j=0; j<3; j++)
```

```
            reflect[i][j] = reflectY[i][j];
```

```
else if(choice == 3)
```

```
    for(int i=0; i<3; i++)
```

```
        for(int j=0; j<3; j++)
```

```
            reflect[i][j] = reflectXY[i][j];
```

```
else if(choice == 4)
```

```

        for(int i=0; i<3; i++)
            for(int j=0; j<3; j++)
                reflect[i][j] = reflectX_Y[i][j];
    else if(choice == 5)
        for(int i=0; i<3; i++)
            for(int j=0; j<3; j++)
                reflect[i][j] = reflect_neg[i][j];
    else
        cout<<"Wrong choice entered";

    cout<<"\nOutput is shown on second screen";

    multiply_matrix(reflect, mat);

    setcolor(CYAN);
    line(result[0][0] +320, 240-result[1][0], 320+result[0][1],240-result[1][1]);
    line(result[0][1] +320, 240-result[1][1], 320+result[0][2],240-result[1][2]);
    line(result[0][2] +320, 240-result[1][2], 320+result[0][0],240-result[1][0]);

}

int main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");

    int i, j;
    cout<<"Enter the co-ordinates of triangle:\n";

```

```

cout<<"Enter x1, x2, x3: ";
for(i=0; i<3; i++)
    cin>>mat[0][i];

cout<<"Enter y1, y2, y3: ";
for(i=0; i<3; i++)
    cin>>mat[1][i];

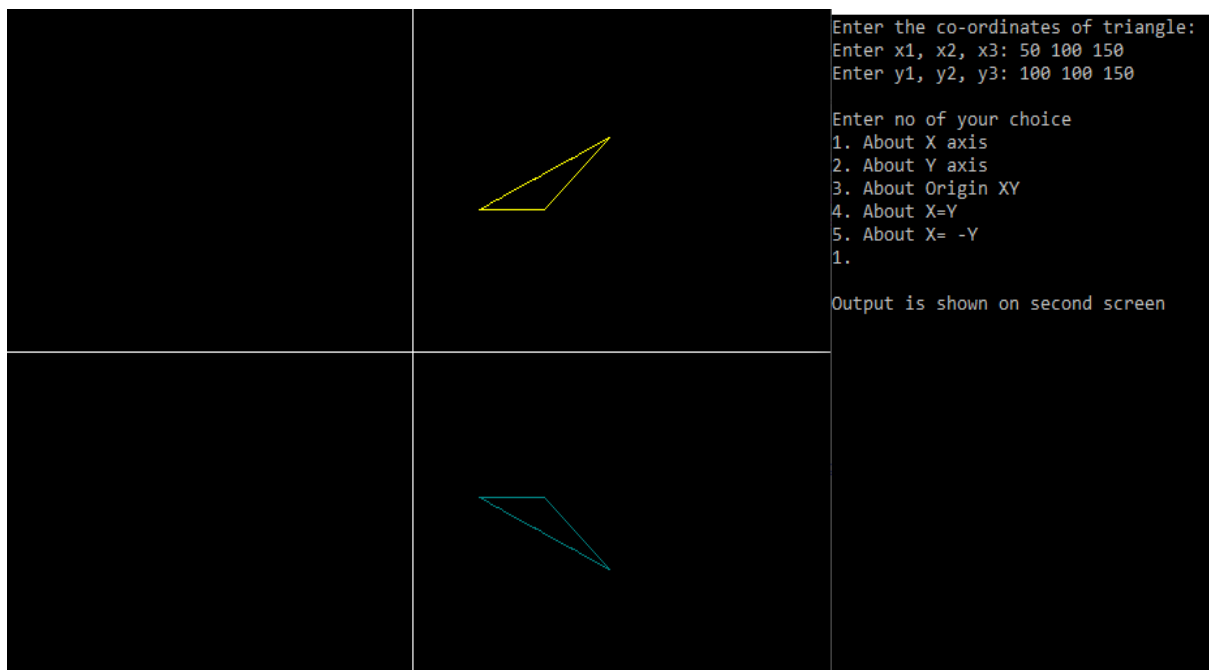
for(i=0; i<3; i++)
    mat[2][i] = 1;



reflection();



getch();
closegraph();
return 0;
}


```

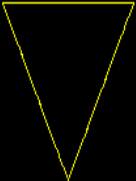
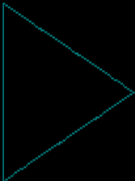
### **OUTPUT:**



		<p>Enter the co-ordinates of triangle:  Enter x1, x2, x3: 50 100 150  Enter y1, y2, y3: 50 100 50</p> <p>Enter no of your choice  1. About X axis  2. About Y axis  3. About Origin XY  4. About X=Y  5. About X= -Y  2.</p> <p>Output is shown on second screen</p>

		<p>Enter the co-ordinates of triangle:  Enter x1, x2, x3: 50 100 150  Enter y1, y2, y3: 100 50 100</p> <p>Enter no of your choice  1. About X axis  2. About Y axis  3. About Origin XY  4. About X=Y  5. About X= -Y  3.</p> <p>Output is shown on second screen</p>
		

		<pre>Enter the co-ordinates of triangle: Enter x1, x2, x3: 100 100 50 Enter y1, y2, y3: 50 100 50  Enter no of your choice 1. About X axis 2. About Y axis 3. About Origin XY 4. About X=Y 5. About X= -Y 4  Output is shown on second screen</pre>

		<pre>Enter the co-ordinates of triangle: Enter x1, x2, x3: 100 150 200 Enter y1, y2, y3: 200 100 200  Enter no of your choice 1. About X axis 2. About Y axis 3. About Origin XY 4. About X=Y 5. About X= -Y 5  Output is shown on second screen</pre>
		

## **PROGRAM -8(b)**

**PROBLEM STATEMENT:** Write a program to implement Reflection of the object.

### **CODE:**

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
int b[3][3]={ 1,1,1,
1,1,1,
0,0,1
};
int c[10][10];
float a[10][10];
int main()
{
    int gm,gd=DETECT,x[10],y[10],nx[10],ny[10],n,midx,midy,i,d,x1,y1,x2,y2;
    float slope;
    int choice;
    initgraph(&gd,&gm,"char*");
    line(getmaxx()/2,0,getmaxx()/2,getmaxy());
    line(0,getmaxy()/2,getmaxx(),getmaxy()/2);
    midx=getmaxx()/2;
    midy=getmaxy()/2;
    printf("Enter the no of sides: ");
    scanf("%d",&n);
    printf("\nEnter the coordinates:\n");
    for(i=1;i<=n;i++)
```

```

{
    printf("P[%d]: ",i);
    scanf("%d%d",&x[i],&y[i]);
}
for(i=1;i<=n;i++)
{
    if(i==n)
    {
        line(x[i]+midx,midy-y[i],x[1]+midx,midy-y[1]);
    }
    else
    {
        line(midx+x[i],midy-y[i],midx+x[i+1],midy-y[i+1]);
    }
}

printf("\n1.About x axis\n2.About y-axis\n3.About origin\n4.About arbitrary
line\nEnter your choice: ");
scanf("%d",&choice);
switch(choice)
{
case 1:
    for(i=1;i<=n;i++)
    {
        nx[i]=x[i];
        ny[i]=-y[i];
    }
    setcolor(YELLOW);
    for(i=1;i<=n;i++)
    {

```

```
    if(i==n)
    {
        line(midx+nx[i],midy-ny[i],midx+nx[1],midy-ny[1]);
    }
    else
    {
        line(midx+nx[i],midy-ny[i],midx+nx[i+1],midy-ny[i+1]);
    }
}
break;
case 2:
for(i=1;i<=n;i++)
{
    nx[i]=-x[i];
    ny[i]=y[i];
}
setcolor(YELLOW);
for(i=1;i<=n;i++)
{
    if(i==n)
    {
        line(midx+nx[i],midy-ny[i],midx+nx[1],midy-ny[1]);
    }
    else
    {
        line(midx+nx[i],midy-ny[i],midx+nx[i+1],midy-ny[i+1]);
    }
}
break;
```



case 3:

```
for(i=1;i<=n;i++)
{
    nx[i]=-x[i];
    ny[i]=-y[i];
}
for(i=1;i<=n;i++)
{
    if(i==n)
    {
        line(midx+nx[i],midy-ny[i],midx+nx[1],midy-ny[1]);
    }
    else
    {
        line(midx+nx[i],midy-ny[i],midx+nx[i+1],midy-ny[i+1]);
    }
}
break;
```

case 4:

```
printf("\nEnter the coordinates of point A: ");
scanf("%d%d",&x1,&y1);
printf("\nEnter the coordinates of point B: ");
scanf("%d%d",&x2,&y2);
slope=((y2-y1)/(x2-x1));
d=y2-(x2*slope);
b[0][0]=(1-(slope*slope))/(1+(slope*slope));
b[0][1]=(2*slope)/((slope*slope)+1);
b[0][2]=(-2*d*slope)/((slope*slope)+1);
b[1][0]=(2*slope)/((slope*slope)+1);
```

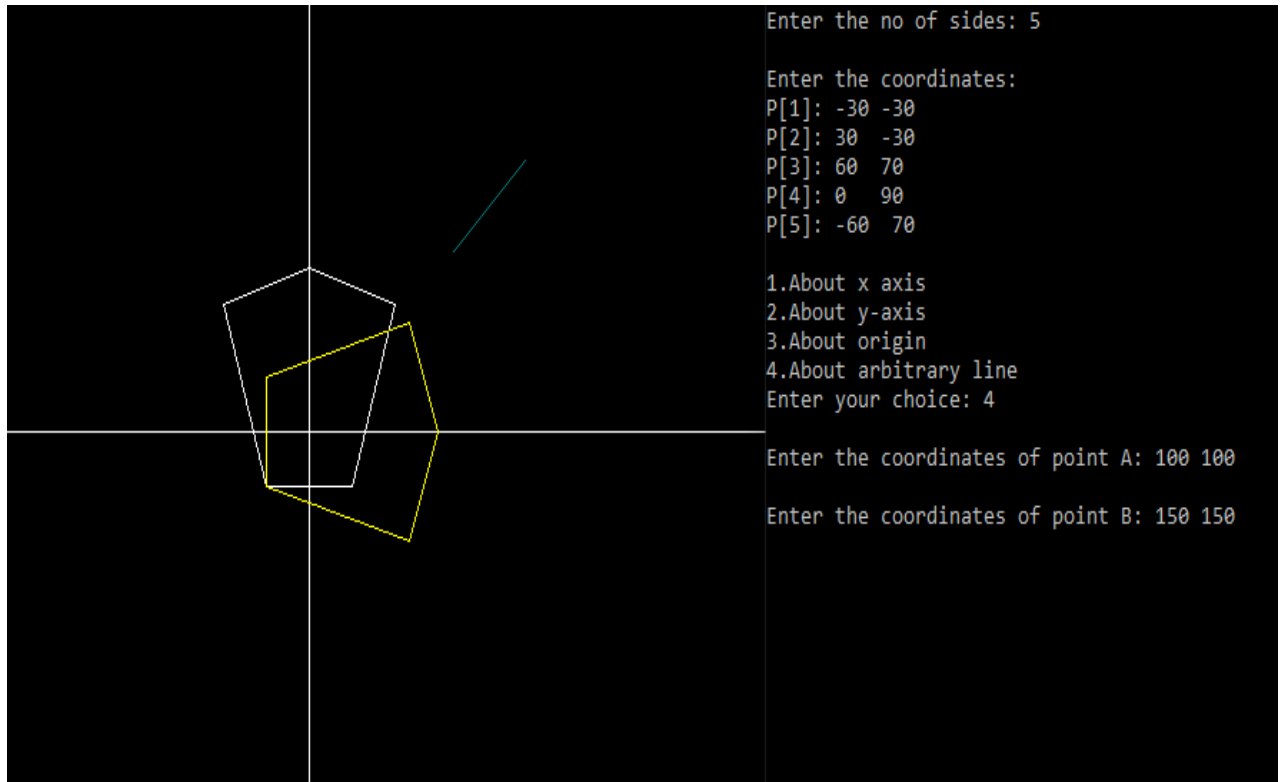
```

b[1][1]=((slope*slope)-1)/((slope*slope)+1);
b[1][2]=(2*d)/((slope*slope)+1);
b[2][0]=0;
b[2][1]=0;
b[2][2]=1;
for(i=1;i<=n;i++)
{
    a[i][0]=x[i];
    a[i][1]=y[i];
    c[i][0]=b[0][0]*a[i][0]+b[0][1]*a[i][1]+b[0][2];
    c[i][1]=b[1][0]*a[i][0]+b[1][1]*a[i][1]+b[1][2];
    //printf("%d%d",c[0][0],c[0][1]);
    x[i]=c[i][0];
    y[i]=c[i][1];
}
setcolor(CYAN);
line(x1+320,240-y1,320+x2,240-y2);
setcolor(YELLOW);
for(i=1;i<=n;i++)
{
    if(i==n)
        line(x[i]+midx,midy-y[i],x[1]+midx,midy-y[1]);
    else
        line(x[i]+midx,midy-y[i],x[i+1]+midx,midy-y[i+1]);
}
break;
default: printf("\nInvalid: ");
}
getch();

```

```
closegraph();  
}
```

### OUTPUT:



## **PROGRAM -9**

**PROBLEM STATEMENT:** Implement shearing of an object.

### **THEORY:**

Shearing deals with changing the shape and size of the 2D object along x-axis and y-axis. It is similar to sliding the layers in one direction to change the shape of the 2D object. It is an ideal technique to change the shape of an existing object in a two dimensional plane. In a two-dimensional plane, the object size can be changed along X direction as well as Y direction.

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

**Shearing in the X-direction:** In this horizontal shearing sliding of layers occur. The homogeneous 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 directions:** Here layers will be slid 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:

$$\begin{bmatrix} 1 & Sh_y & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

### **CODE:**

```
#include<iostream>

#include<conio.h>

#include<graphics.h>

#include<math.h>

using namespace std;

void disp(int n,float c[][3]){

float maxx,maxy;

int i;
```

```

maxx=getmaxx();
maxy=getmaxy();
maxx=maxx/2;
maxy=maxy/2;
i=0;
while(i<n-1){
line(maxx+c[i][0],maxy-c[i][1],maxx+c[i+1][0],maxy-c[i+1][1]);
i++; }
i=n-1;
line(maxx+c[i][0],maxy-c[i][1],maxx+c[0][0],maxy-c[0][1]);
setcolor(GREEN);
line(0,maxy,maxx*2,maxy);
line(maxx,0,maxx,maxy*2);
setcolor(WHITE);}

void mul(int n,float b[][3],float c[][3],float a[][3]){
int i,j,k;
for(i=0;i<n;i++)
for(j=0;j<3;j++)
a[i][j]=0;
for(i=0;i<n;i++)
for(j=0;j<3;j++)
for(k=0;k<3;k++) {
a[i][j]=a[i][j]+(c[i][k]*b[k][j]); } }

void translation(int n,float c[][3],float tx,float ty){
int i;
for(i=0;i<n;i++) {
c[i][0]=c[i][0]+tx;
c[i][1]=c[i][1]+ty; }
}

void scaling(int n,float c[][3],float sx,float sy){

```

```

float b[10][3],a[10][3];

int i,j;
for(j=0;j<3;j++)
b[i][j]=0;
b[0][0]=sx;
b[1][1]=sy;
b[2][2]=1;
mul(n,b,c,a);
setcolor(RED);
disp(n,a);}

void rotation(int n,float c[][3],float ra){
int i=0,j;
float b[10][3],xp,yp,a[10][3];
xp=c[0][0];
yp=c[0][1];
for(i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
b[0][0]=b[1][1]=cos(ra*3.14/180);
b[0][1]=sin(ra*3.14/180);
b[1][0]=-sin(ra*3.14/180);;
b[2][0]=(-xp*cos(ra*3.14/180))+(yp*sin(ra*3.14/180))+xp;
b[2][1]=(-xp*sin(ra*3.14/180))-(yp*cos(ra*3.14/180))+yp;
b[2][2]=1;
mul(n,b,c,a);
setcolor(RED);
disp(n,a);}

void refthx(int n,float c[][3]){
int i=0,j;
float b[10][3],a[10][3];

```

```
for(i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
b[0][0]=b[2][2]=1;
b[1][1]=-1;
mul(n,b,c,a);
setcolor(RED);
disp(n,a);}

void refthy(int n,float c[][3]){
int i=0,j;
float b[10][3],a[10][3];
for(i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
b[1][1]=b[2][2]=1;
b[0][0]=-1;
mul(n,b,c,a);
setcolor(RED);
disp(n,a);}

void reforg(int n,float c[][3]){
int i=0,j;
float b[10][3],a[10][3];
for(i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
b[0][0]=b[1][1]=-1;
b[2][2]=1;
mul(n,b,c,a);
setcolor(RED);
disp(n,a);}
```

```

void refthyx(int n,float c[][3]){
int i=0,j;
float b[10][3],a[10][3];
for(i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
b[0][1]=b[1][0]=b[2][2]=1;
mul(n,b,c,a);
setcolor(RED);
disp(n,a);}

void refthynegx(int n,float c[][3]){
int i=0,j;
float b[10][3],a[10][3];
for(i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
b[0][1]=b[1][0]=-1;
b[2][2]=1;
mul(n,b,c,a);
setcolor(RED);
disp(n,a);}

void shearx(int n,float c[][3],float shx){
int i=0,j;
float b[10][3],a[10][3];
for(i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
b[0][0]=b[1][1]=b[2][2]=1;
b[1][0]=shx;
mul(n,b,c,a);

```



```

setcolor(RED);
disp(n,a);}
void sheary(int n,float c[][3],float shy){
int i=0,j;
float b[10][3],a[10][3];
for(i=0;i<3;i++)
for(j=0;j<3;j++)
b[i][j]=0;
b[0][0]=b[1][1]=b[2][2]=1;
b[0][1]=shy;
mul(n,b,c,a);
setcolor(RED);
disp(n,a);}
int main(){
int i,j,k,cho,n,gd=DETECT,gm;
float c[10][3],tx,ty,sx,sy,ra;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
cout<<"Enter vertices ";
cin>>n;
for (i=0;i<n;i++) {
cout<<"Enter coordinates ",i+1;
cin>>c[i][0]>>c[i][1];
c[i][2]=1; }
do {
cleardevice();
cout<<"\n MENU" ;
cout<<"\n 1.Translation" ;
cout<<"\n 2.Scaling" ;
cout<<"\n 3.Rotation" ;
cout<<"\n 4.Shear";

```

```
cout<<"\n Enter your Choice";
cin>>cho;
switch(cho) {
case 1:
cout<<"\n Enter translation factor for X and Y axis: ";
cin>>tx>>ty;
cleardevice();
setcolor(15);
disp(n,c);
translation(n,c,tx,ty);
setcolor(15);
disp(n,c);
getch();
break;
case 2:
cout<<"\n Enter scaling factor for X and Y axis:";
cin>>sx>>sy;
cleardevice();
setcolor(15);
disp(n,c);
scaling(n,c,sx,sy);
setcolor(15);
disp(n,c);
getch();
break;
case 3:
cout<<"\n Enter rotation factor:";
cin>>ra;
cleardevice();
disp(n,c);
```

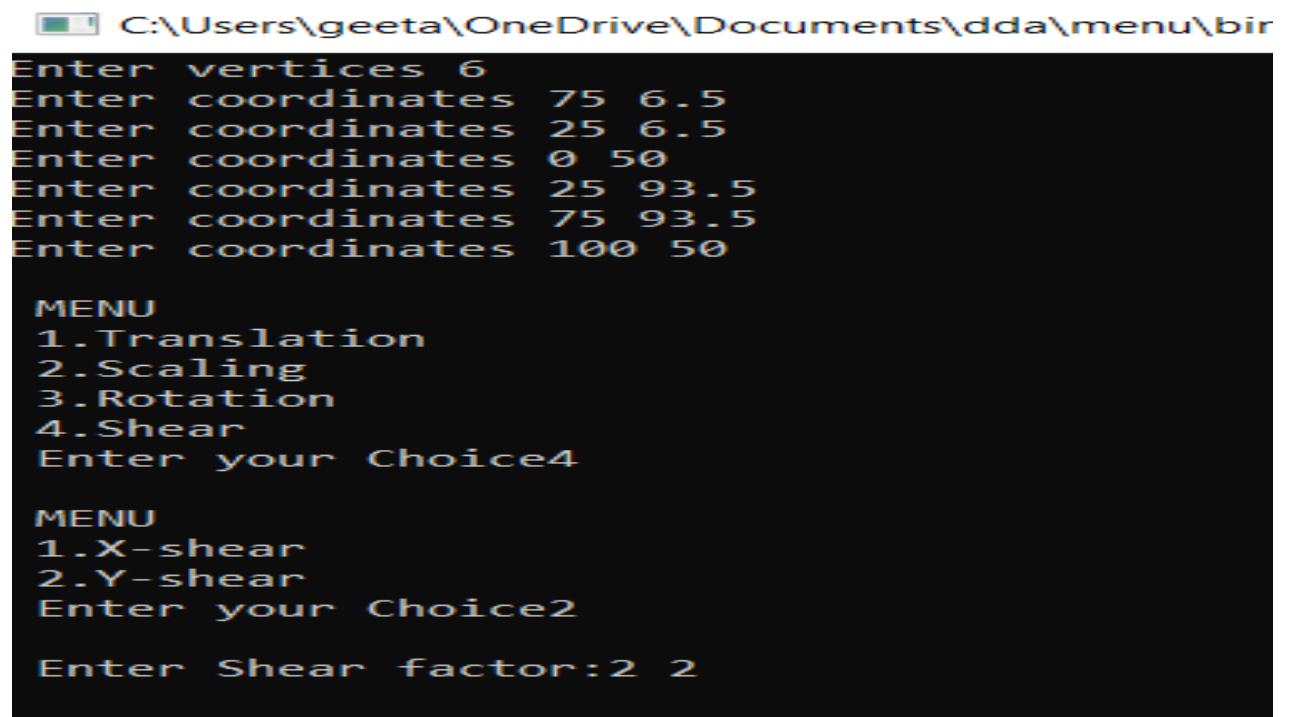
```
rotation(n,c,ra);
getch();
break;
while(cho!=5);
case 4:
int cha;
float shx,shy;
do {
cleardevice();
cout<<"\n MENU";
cout<<"\n 1.X-shear";
cout<<"\n 2.Y-shear";
cout<<"\n Enter your Choice";
cin>>cha;
switch(cha) {
case 1:
cout<<"\n Enter Shear factor:";
cin>>shx;
cleardevice();
setcolor(15);
disp(n,c);
shearx(n,c,shx);
getch();
break;
case 2:
cout<<"\n Enter Shear factor:";
cin>>shy;
cleardevice();
setcolor(15);
disp(n,c);
```

```

shearx(n,c,shy);
getch();
break;
default:
cout<<"\n Invalid";
break; } }
while(cho!=2);
break;
default:
cout<<"\n Invalid";
break; } }
while(cho!=6);
getch();
closegraph();}

```

## **OUTPUT:**



```

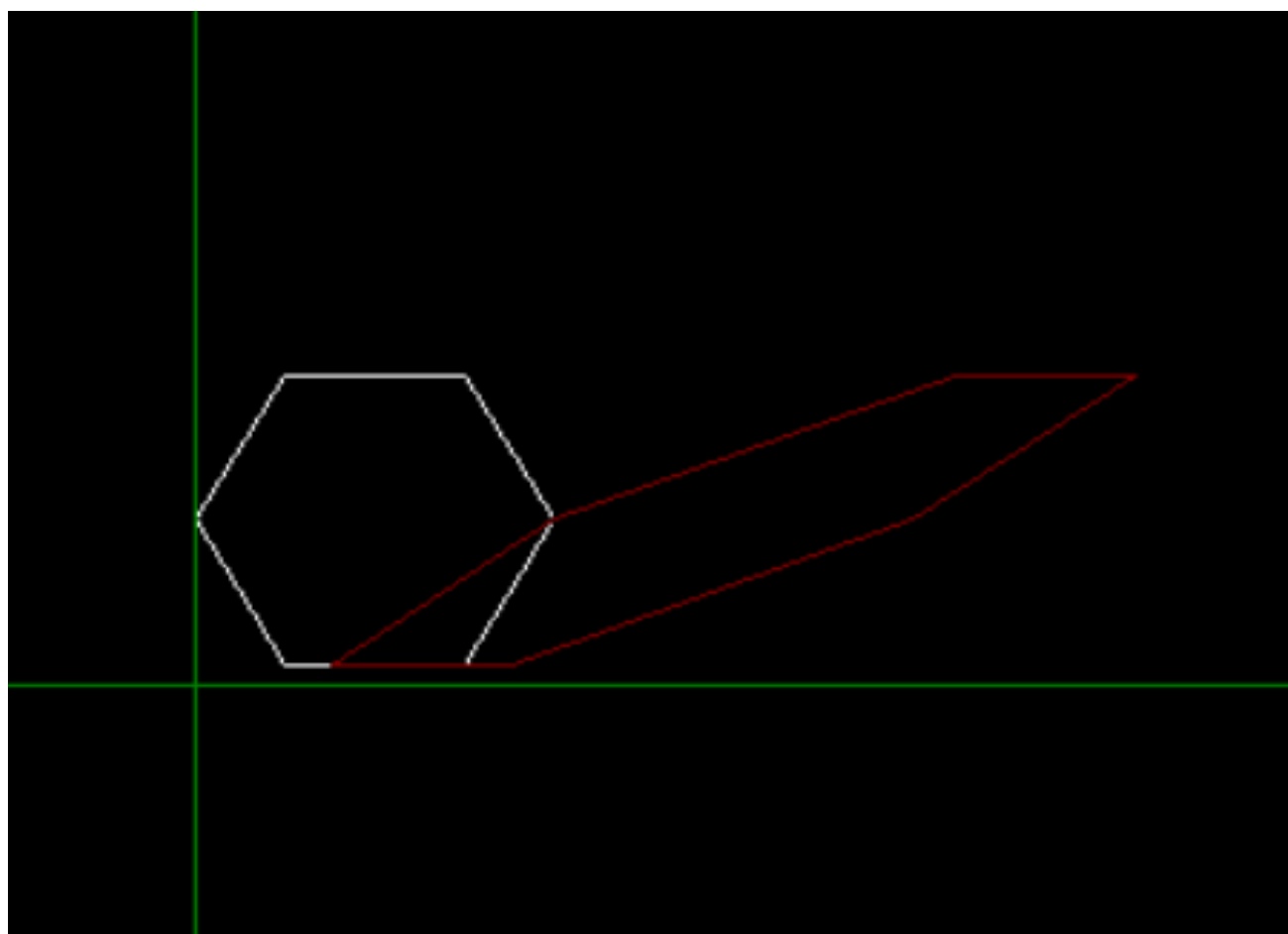
C:\Users\geeta\OneDrive\Documents\dda\menu\bir
Enter vertices 6
Enter coordinates 75 6.5
Enter coordinates 25 6.5
Enter coordinates 0 50
Enter coordinates 25 93.5
Enter coordinates 75 93.5
Enter coordinates 100 50

MENU
1.Translation
2.Scaling
3.Rotation
4.Shear
Enter your Choice4

MENU
1.X-shear
2.Y-shear
Enter your Choice2

Enter Shear factor:2 2

```



## **PROGRAM - 10**

**PROBLEM STATEMENT:** Write a program to implement window to viewport.

### **THEORY:**

Once object description has been transmitted to the viewing reference frame, we choose the window extends in viewing coordinates and selects the viewport limits in normalized coordinates.

Object descriptions are then transferred to normalized device coordinates:

We do this thing using a transformation that maintains the same relative placement of an object in normalized space as they had in viewing coordinates.

If a coordinate position is at the center of the viewing window:

It will display at the center of the viewport.

In order to maintain the same relative placement of the point in the viewport as in the window, we require:

$$\frac{x_v - x_{v_{\min}}}{x_{v_{\max}} - x_{v_{\min}}} = \frac{x_w - x_{w_{\min}}}{x_{w_{\max}} - x_{w_{\min}}} \dots\dots\dots \text{equation 1}$$

$$\frac{y_v - y_{v_{\min}}}{y_{v_{\max}} - y_{v_{\min}}} = \frac{y_w - y_{w_{\min}}}{y_{w_{\max}} - y_{w_{\min}}}$$

Solving these impressions for the viewport position (xv, yv), we have

$$\begin{aligned} x_v &= x_{v_{\min}} + (x_w - x_{w_{\min}})S_x \\ y_v &= y_{v_{\min}} + (y_w - y_{w_{\min}})S_y \dots\dots\dots \text{equation 2} \end{aligned}$$

Where scaling factors are

$$S_x = \frac{x_{v_{\max}} - x_{v_{\min}}}{x_{w_{\max}} - x_{w_{\min}}}$$

$$S_y = \frac{y_{v_{\max}} - y_{v_{\min}}}{y_{w_{\max}} - y_{w_{\min}}}$$

## CODE:

```
#include<graphics.h>
#include<conio.h>
#include<stdio.h>
int main()
{
int W_xmax,W_ymax,W_xmin,W_ymin;
int V_xmax,V_ymax,V_xmin,V_ymin;
float sx,sy;
int x,x1,x2,y,y1,y2;
int gr=DETECT,gm;
initgraph(&gr,&gm,"C:\\TURBOC3\\BGI");
printf("\n***** Window to Viewport *****\n");
printf("Enter the coordinates for triangle \n x and y = ");
scanf("%d %d",&x,&y);
printf("\n x1 and y1 = ");
scanf("%d %d",&x1,&y1);
printf("\n x2 and y2 = ");
scanf("%d %d",&x2,&y2);

printf("Please enter Window coordinates \n First enter XMax, YMax =");
scanf("%d %d",&W_xmax,&W_ymax);
printf("\n Now, enter XMin, YMin =");
scanf("%d %d",&W_xmin,&W_ymin);
cleardevice();
delay(50);
//Window
rectangle(W_xmin,W_ymin ,W_xmax,W_ymax);
outtextxy(W_xmin,W_ymin-10, "Window");
//drawing a triangle
```

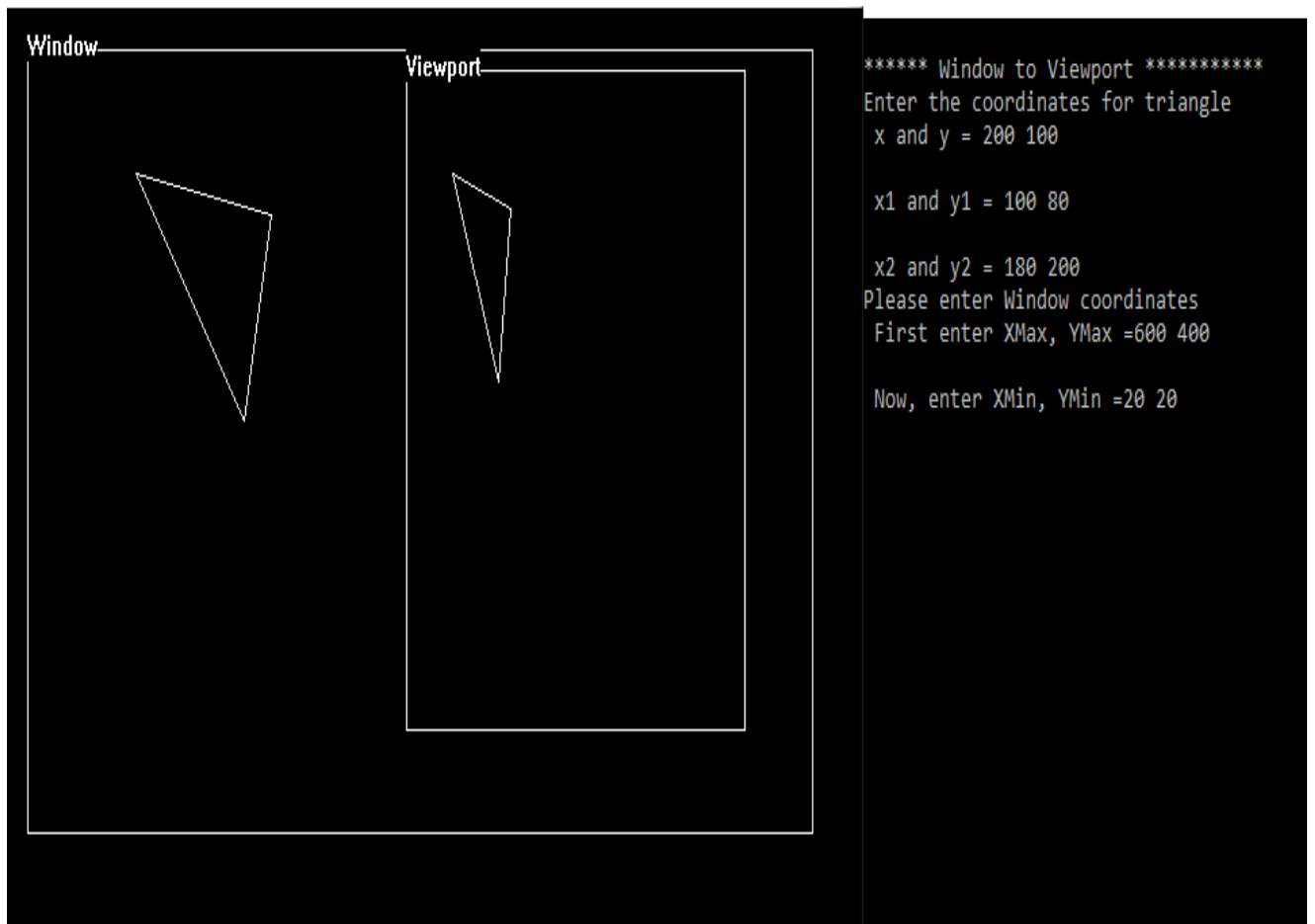
```

line(x,y,x1,y1);
line(x1,y1,x2,y2);
line(x2,y2,x,y);
// viewport
V_xmin=300;
V_ymin=30;
V_xmax=550;
V_ymax=350;
rectangle(V_xmin,V_ymin,V_xmax,V_ymax);
outtextxy(V_xmin,V_ymin-10, "Viewport");
// calculatng Sx and Sy
sx=(float)(V_xmax-V_xmin)/(W_xmax - W_xmin);
sy=(float)(V_ymax-V_ymin)/(W_ymax - W_ymin);
x = V_xmin +(float)((x-W_xmin)*sx) ;
x1= V_xmin +(float)((x1-W_xmin)*sx);
x2= V_xmin +(float)((x2-W_xmin)*sx) ;
y = V_ymin +(float)((y- W_ymin)*sy) ;
y1= V_ymin +(float)((y1-W_ymin)*sy) ;
y2= V_ymin +(float)((y2-W_ymin)*sy);
// drawing triangle
line(x,y,x1,y1);
line(x1,y1,x2,y2);
line(x2,y2,x,y);
getch();
closegraph();
}

```



## OUTPUT:



## **PROGRAM - 11**

**PROBLEM STATEMENT:** Write a program to implement cohen Sutherland line clipping algorithm

### **THEORY:**

In the algorithm, first of all, it is detected whether line lies inside the screen or it is outside the screen. All lines come under any one of the following categories:

1. Visible
2. Not Visible
3. Clipping Case

**1. Visible:** If a line lies within the window, i.e., both endpoints of the line lies within the window. A line is visible and will be displayed as it is.

**2. Not Visible:** If a line lies outside the window it will be invisible and rejected. Such lines will not display. If any one of the following inequalities is satisfied, then the line is considered invisible. Let A ( $x_1, y_1$ ) and B ( $x_2, y_2$ ) are endpoints of line.

$x_{min}, x_{max}$  are coordinates of the window.

$y_{min}, y_{max}$  are also coordinates of the window.

$x_1 > x_{max}$

$x_2 > x_{max}$

$y_1 > y_{max}$

$y_2 > y_{max}$

$x_1 < x_{min}$

$x_2 < x_{min}$

$y_1 < y_{min}$

$y_2 < y_{min}$

**3. Clipping Case:** If the line is neither visible case nor invisible case. It is considered to be clipped case. First of all, the category of a line is found based on nine regions given below. All nine regions are assigned codes. Each code is of 4 bits. If both endpoints of the line have end bits zero, then the line is considered to be visible.

### **CODE:**

```
#include <iostream>
```

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

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

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

```
#include<math.h>
```

```
int main()
```

```

{
int rcode_begin[4]={0,0,0,0},rcode_end[4]={0,0,0,0},region_code[4];
int W_xmax,W_ymax,W_xmin,W_ymin,flag=0;
float slope;
int x,y,x1,y1,i, xc,yc;
int gr=DETECT,gm;
initgraph(&gr,&gm,"C:\\TURBOC3\\BGI");
//printf("\n*** Cohen Sutherland Line Clipping algorithm ****");
printf("\n Enter XMin,YMin,XMax,YMax-");

scanf("%d %d",&W_xmin,&W_ymin);
scanf("%d %d",&W_xmax,&W_ymax);
printf("\n Enter initial point x and y-");
scanf("%d %d",&x,&y);
printf("\n Enter final point x1 and y1-");
scanf("%d %d",&x1,&y1);
cleardevice();
rectangle(W_xmin,W_ymin,W_xmax,W_ymax);
line(x,y,x1,y1);
line(0,0,600,0);
line(0,0,0,600);
if(y>W_ymax) {
rcode_begin[0]=1;    // Top
flag=1 ;
}
if(y<W_ymin) {
rcode_begin[1]=1;    // Bottom
flag=1;
}
if(x>W_xmax) {

```

```

rcode_begin[2]=1;      // Right
flag=1;
}
if(x<W_xmin) {
rcode_begin[3]=1;      //Left
flag=1;
}

//end point of Line
if(y1>W_ymax){
rcode_end[0]=1;        // Top
flag=1;
}
if(y1<W_ymin) {
rcode_end[1]=1;        // Bottom
flag=1;
}
if(x1>W_xmax){
rcode_end[2]=1;        // Right
flag=1;
}
if(x1<W_xmin){
rcode_end[3]=1;        //Left
flag=1;
}
if(flag==0)
{
printf("Already inside the window");
}
flag=1;

```

```

for(i=0;i<4;i++){
region_code[i]= rcode_begin[i] && rcode_end[i] ;
if(region_code[i]==1)
    flag=0;
}
if(flag==0)
{
printf("\n Outside the window");
}
else{
slope=(float)(y1-y)/(x1-x);
if(rcode_begin[2]==0 && rcode_begin[3]==1) //left
{
y=y+(float) (W_xmin-x)*slope ;
x=W_xmin;

}
if(rcode_begin[2]==1 && rcode_begin[3]==0) // right
{
y=y+(float) (W_xmax-x)*slope ;
x=W_xmax;

}
if(rcode_begin[0]==1 && rcode_begin[1]==0) // top
{
x=x+(float) (W_ymax-y)/slope ;
y=W_ymax;

}
if(rcode_begin[0]==0 && rcode_begin[1]==1) // bottom

```

```

{
x=x+(float) (W_ymin-y)/slope ;
y=W_ymin;

}

// end points
if(rcode_end[2]==0 && rcode_end[3]==1) //left
{
y1=y1+(float) (W_xmin-x1)*slope ;
x1=W_xmin;

}

if(rcode_end[2]==1 && rcode_end[3]==0) // right
{
y1=y1+(float) (W_xmax-x1)*slope ;
x1=W_xmax;

}

if(rcode_end[0]==1 && rcode_end[1]==0) // top
{
x1=x1+(float) (W_ymax-y1)/slope ;
y1=W_ymax;

}

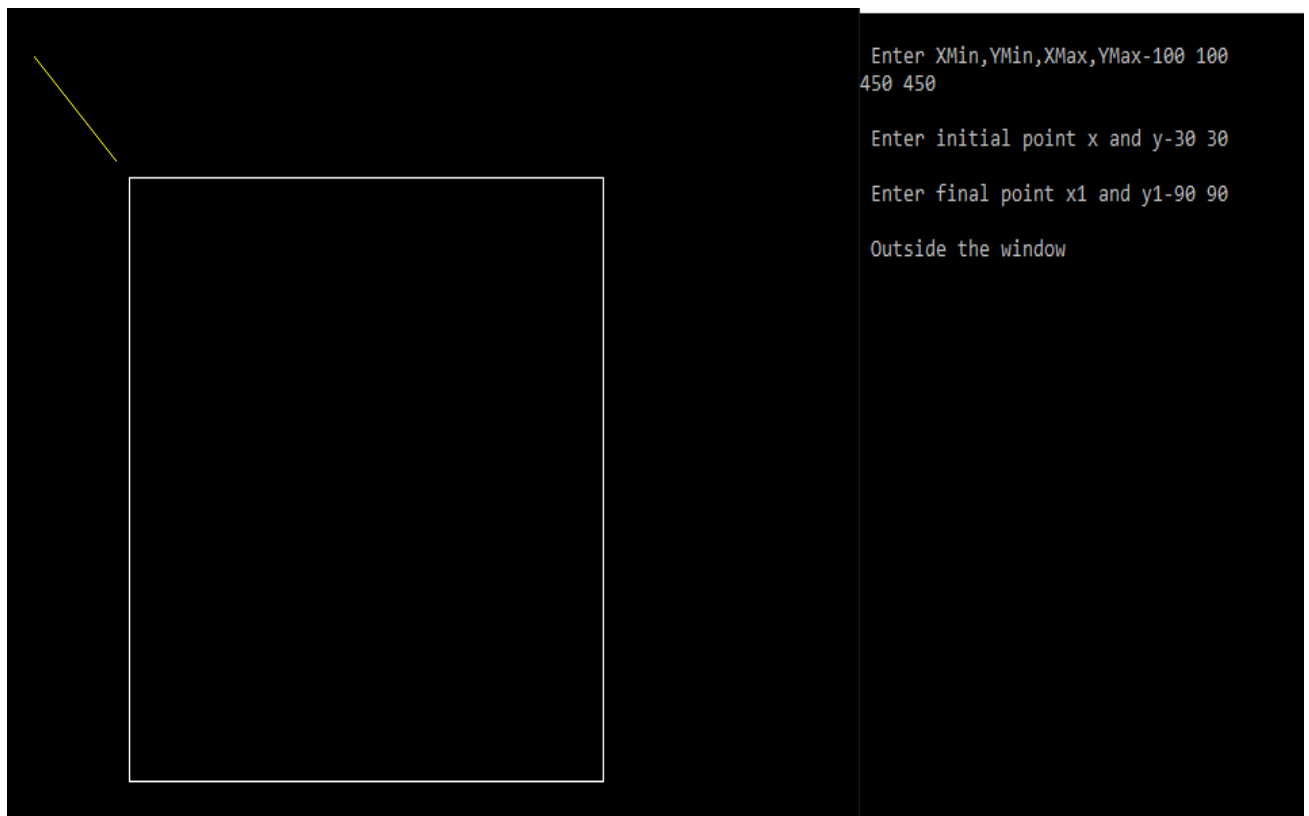
if(rcode_end[0]==0 && rcode_end[1]==1) // bottom
{
x1=x1+(float) (W_ymin-y1)/slope ;
y1=W_ymin;

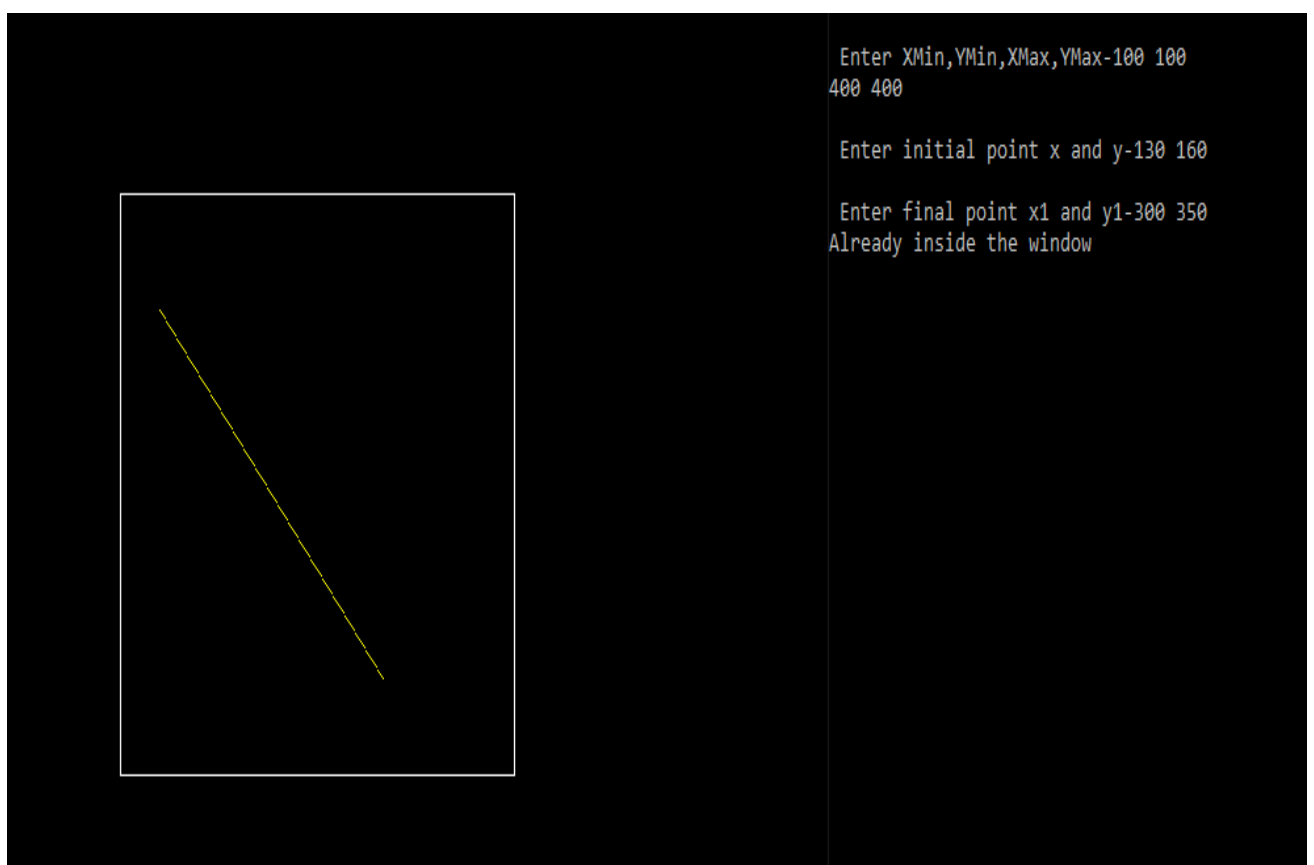
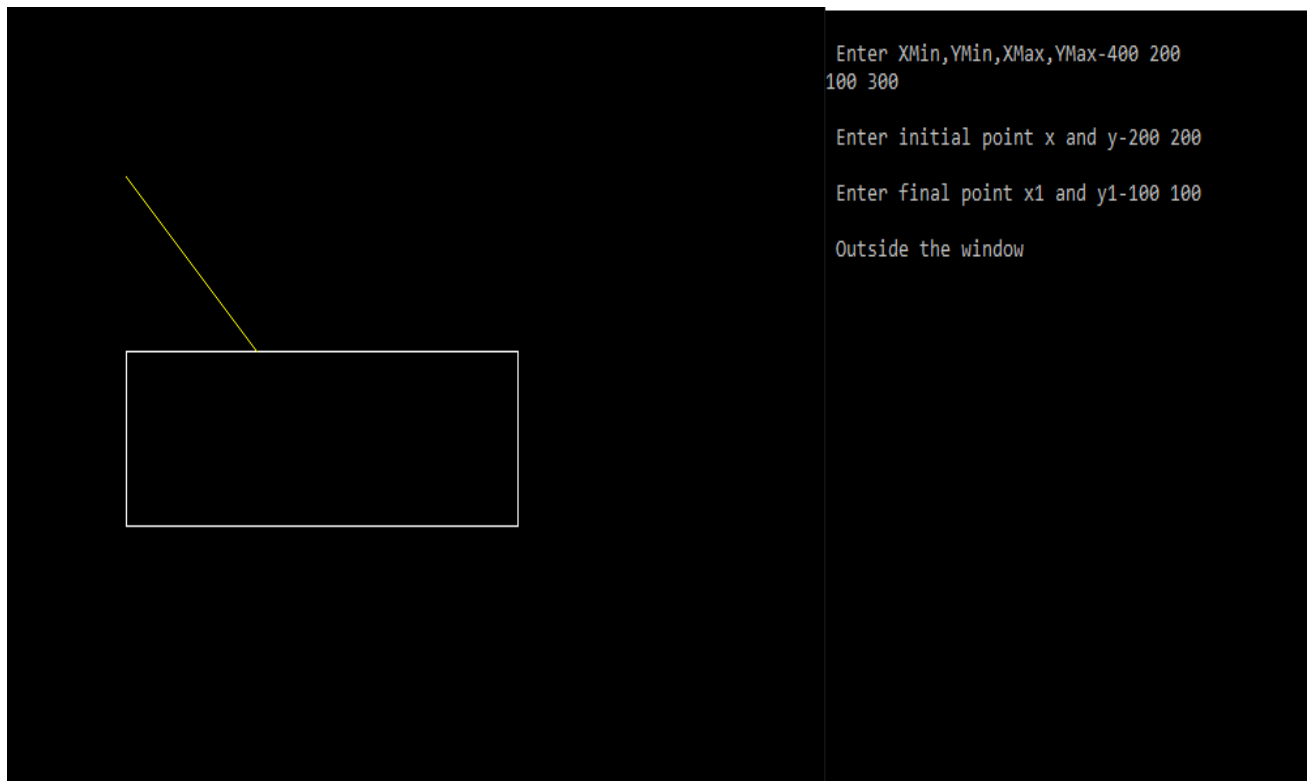
}

```

```
}  
delay(1000);  
clearviewport();  
rectangle(W_xmin,W_ymin,W_xmax,W_ymax);  
line(0,0,600,0);  
line(0,0,0,600);  
setcolor(YELLOW);  
delay(100);  
line(x,y,x1,y1);  
getch();  
closegraph();  
return 0;  
}
```

### **OUTPUT:**







## **PROGRAM - 12**

**PROBLEM STATEMENT:** Write a program to implement Sutherland - Hodgeman polygon clipping algorithm

### **THEORY:**

It is performed by processing the boundary of polygon against each window corner or edge. First of all entire polygon is clipped against one edge, then resulting polygon is considered, then the polygon is considered against the second edge, so on for all four edges.

#### **Four possible situations while processing**

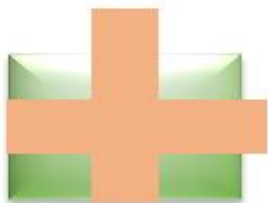
1. If the first vertex is an outside the window, the second vertex is inside the window. Then second vertex is added to the output list. The point of intersection of window boundary and polygon side (edge) is also added to the output line.
2. If both vertexes are inside window boundary. Then only second vertex is added to the output list.
3. If the first vertex is inside the window and second is an outside window. The edge which intersects with window is added to output list.
4. If both vertices are the outside window, then nothing is added to output list.



**Original polygon**



**Clipping against left edge of window**



**Clipping against bottom corner of window**



**Clipping against Top of window**



**Clipping against right edge of window**

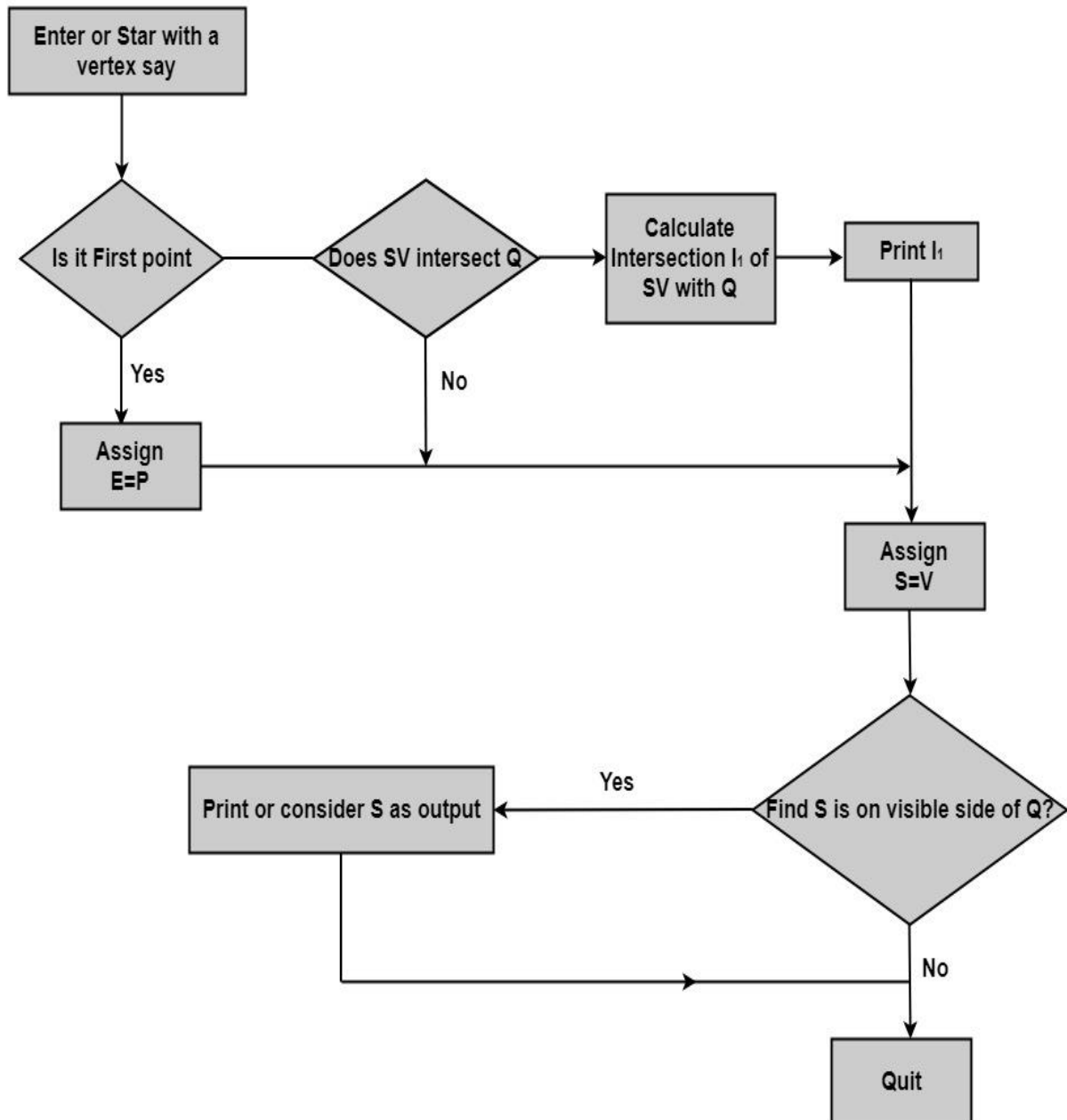


**Final polygon after Clipping**

## Disadvantage of Cohen Hodgmen Algorithm:

This method requires a considerable amount of memory. The first of all polygons are stored in original form. Then clipping against left edge done and output is stored. Then clipping against right edge done, then top edge. Finally, the bottom edge is clipped. Results of all these operations are stored in memory. So wastage of memory for storing intermediate polygons.

## Sutherland Hodgemen Algorithm:



## **CODE:**

```
#include<iostream>
#include<conio.h>
#include<graphics.h>
#define round(a) ((int)(a+0.5))
using namespace std;
int k;
float xmin,ymin,xmax,ymax,arr[20],m;
void clipl(float x1,float y1,float x2,float 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(float x1,float y1,float x2,float 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(float x1,float y1,float x2,float 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(float x1,float y1,float x2,float 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;
}
}

int main()
{
    int gdriver=DETECT,gmode,n,poly[20],i;
    float xi,yi,xf,yf,polyy[20];

    cout<<"Coordinates of rectangular clip window :\\nxmin,ymin      :";
    cin>>xmin>>ymin;
    cout<<"xmax,ymax      :";
    cin>>xmax>>ymax;
    cout<<"\\n\\nPolygon to be clipped :\\nNumber of sides      :";
    cin>>n;
    cout<<"Enter the coordinates :";
    for(i=0;i < 2*n;i++)
        cin>>polyy[i];
    polyy[i]=polyy[0];
    polyy[i+1]=polyy[1];
    for(i=0;i < 2*n+2;i++)
        poly[i]=round(polyy[i]);
    initgraph(&gdriver,&gmode,"C:\\TC\\BGI");
    setcolor(RED);

```

```

rectangle(xmin,ymax,xmax,ymin);
cout<<"\nUNCLIPPED POLYGON IS SHOWN";
setcolor(MAGENTA);
fillpoly(n,poly);

//LINE MARKED -----PRESS ANY KEY after selecting WINDOW BGI
    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++)
    polyyy[i]=arr[i];
polyy[i]=polyyy[0];
polyy[i+1]=polyyy[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++)
    polyyy[i]=arr[i];
polyy[i]=polyyy[0];
polyy[i+1]=polyyy[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++)
    polyyy[i]=arr[i];
polyy[i]=polyyy[0];
polyy[i+1]=polyyy[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]=round(arr[i]);
if(k)
    fillpoly(k/2,poly);
setcolor(white);
rectangle(xmin,ymax,xmax,ymin);
cout<<"\nCLIPPED POLYGON IS SHOWN NOW";

getch();
closegraph();
return 0;
}

```

### **OUTPUT:**

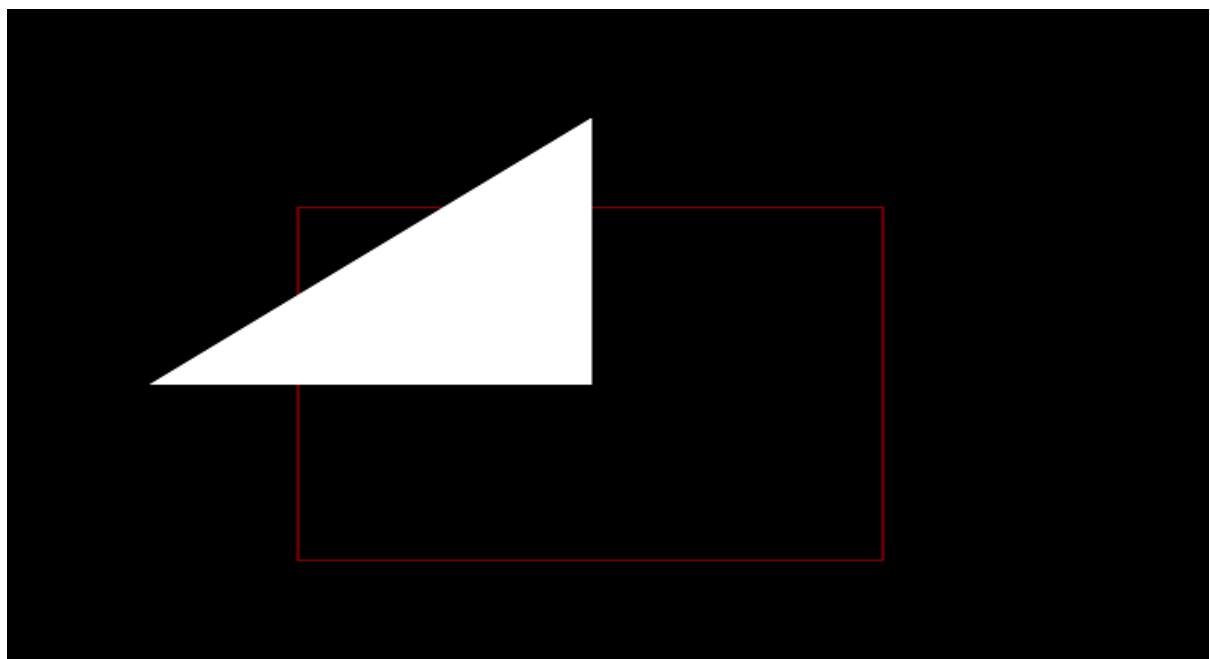
```

Coordinates of rectangular clip window :
xmin,ymin          :200 200
xmax,ymax          :400 400

Polygon to be clipped :
Number of sides     :3
Enter the coordinates :
150 300
300 150
300 300

UNCLIPPED POLYGON IS SHOWN

```

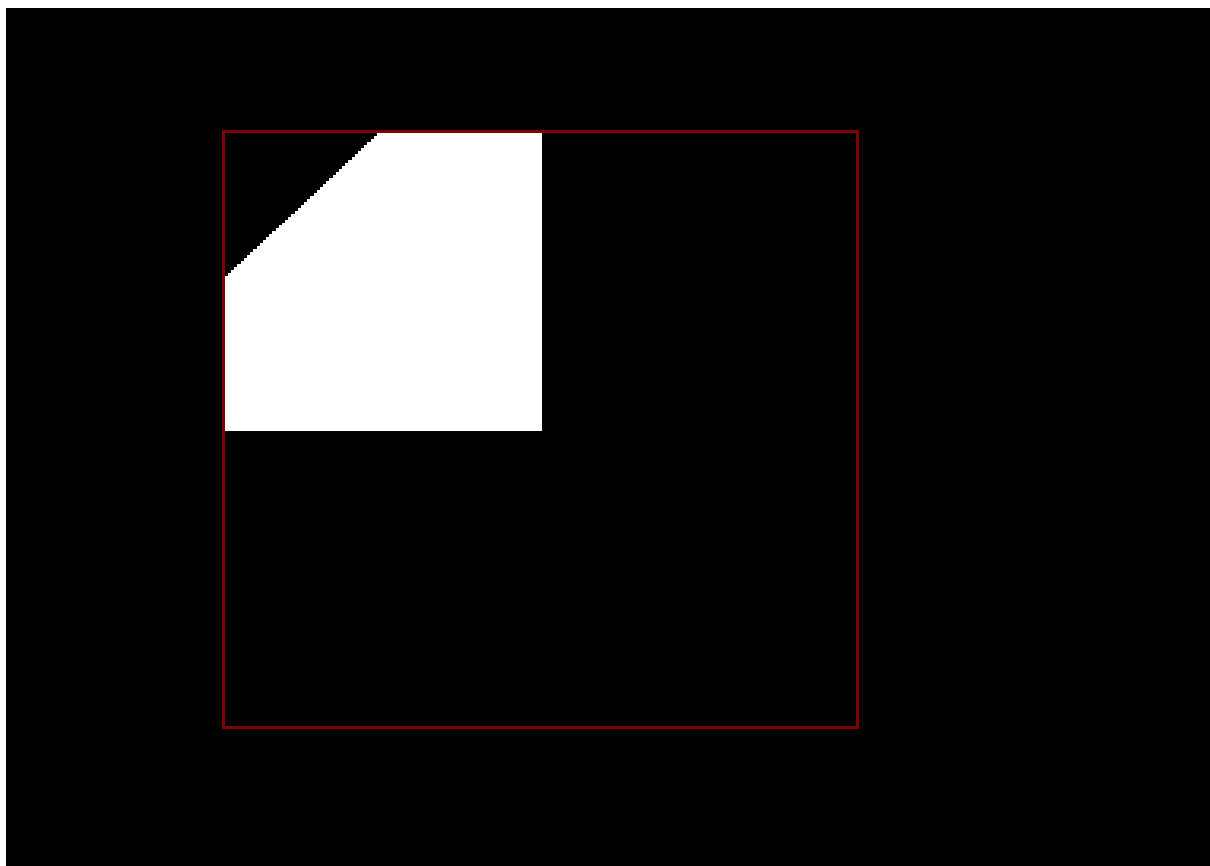




```
Coordinates of rectangular clip window :  
xmin,ymin          :200 200  
xmax,ymax          :400 400
```

```
Polygon to be clipped :  
Number of sides      :3  
Enter the coordinates :  
150 300  
300 150  
300 300
```

```
UNCLIPPED POLYGON IS SHOWN  
CLIPPED POLYGON IS SHOWN NOW
```



## PROGRAM – 13(Extra work)

**PROBLEM STATEMENT:** Write a program to implement

a. **Bezier curve**

b. **Hyperbola**

**THEORY:** Bezier curve was founded by a French scientist named Pierre Bézier. This Curve is drawn by using Control points. In this, Approximate tangents act as control points which are used to generate the desired Bezier. It is a parametric curve which follows bernstein polynomial as the basis function.

$B(t) = \sum_{k=0}^n P_k B_{kn}(t)$  , Where t lies between 0 and 1,  $0 \leq t \leq 1$

$P_k$  represents number of control points

$$B_{k,n}(t) = {}^n C_k u^k (1-t)^{n-k}$$

### Types of Bezier Curve

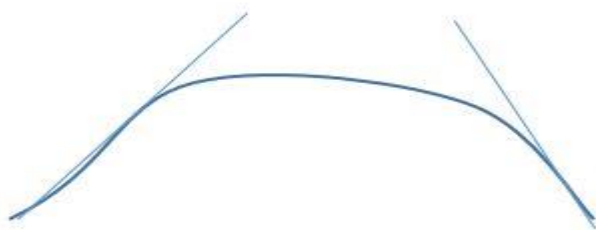
1) Simple Bezier Curve : The simple line connecting endpoint.



2) Quadric Bezier Curve: Quadric curve using 3 control points



3) Cubic Bezier Curve: Cubic curve using 4 control points



## **Properties:**

- 1) A Bezier curve always depends on the number of control points that require to draw it.
- 2) Curve can be drawn using endpoints only.
- 3) The polynomial equation also depends on the number of control points. Suppose,  $n$  is a control point then the degree of the polynomial equation will be  $n-1$ .
- 4) Curve passes through initial and terminating control points.
- 5) Closed Bezier curve can be generated by making the first and last control points the same.

## **CODE:**

```
#include<graphics.h>

#include<math.h>

#include<conio.h>

#include<stdio.h>

int main()

{

int x[15],y[15],i,n;

double put_x,put_y,t;

int gr=DETECT,gm;

initgraph(&gr,&gm,"C:\\TURBOC3\\BGI");

int xp=getmaxx()/2;

int yp=getmaxy()/2;

line(getmaxx()/2,0,getmaxx()/2,getmaxy());

line(0,getmaxy()/2,getmaxx(),getmaxy()/2);
```

```

printf("\n***** Bezier Curve *****");

printf("\nNumber of control points:");

scanf("%d",&n);

printf("\n Please enter x and y coordinates ");

for(i=0;i<4;i++)

{

scanf("%d%d",&x[i],&y[i]);

putpixel(x[i],y[i],3);          // Control Points

}


for(t=0.0;t<=1.0;t=t+0.001)      // t always lies between 0 and 1

{

put_x = pow(1-t,3)*x[0] + 3*t*pow(1-t,2)*x[1] + 3*t*t*(1-t)*x[2] + pow(t,3)*x[3]; //
Formula to draw curve

put_y = pow(1-t,3)*y[0] + 3*t*pow(1-t,2)*y[1] + 3*t*t*(1-t)*y[2] + pow(t,3)*y[3];

putpixel(put_x,put_y, YELLOW);    // putting pixel

}

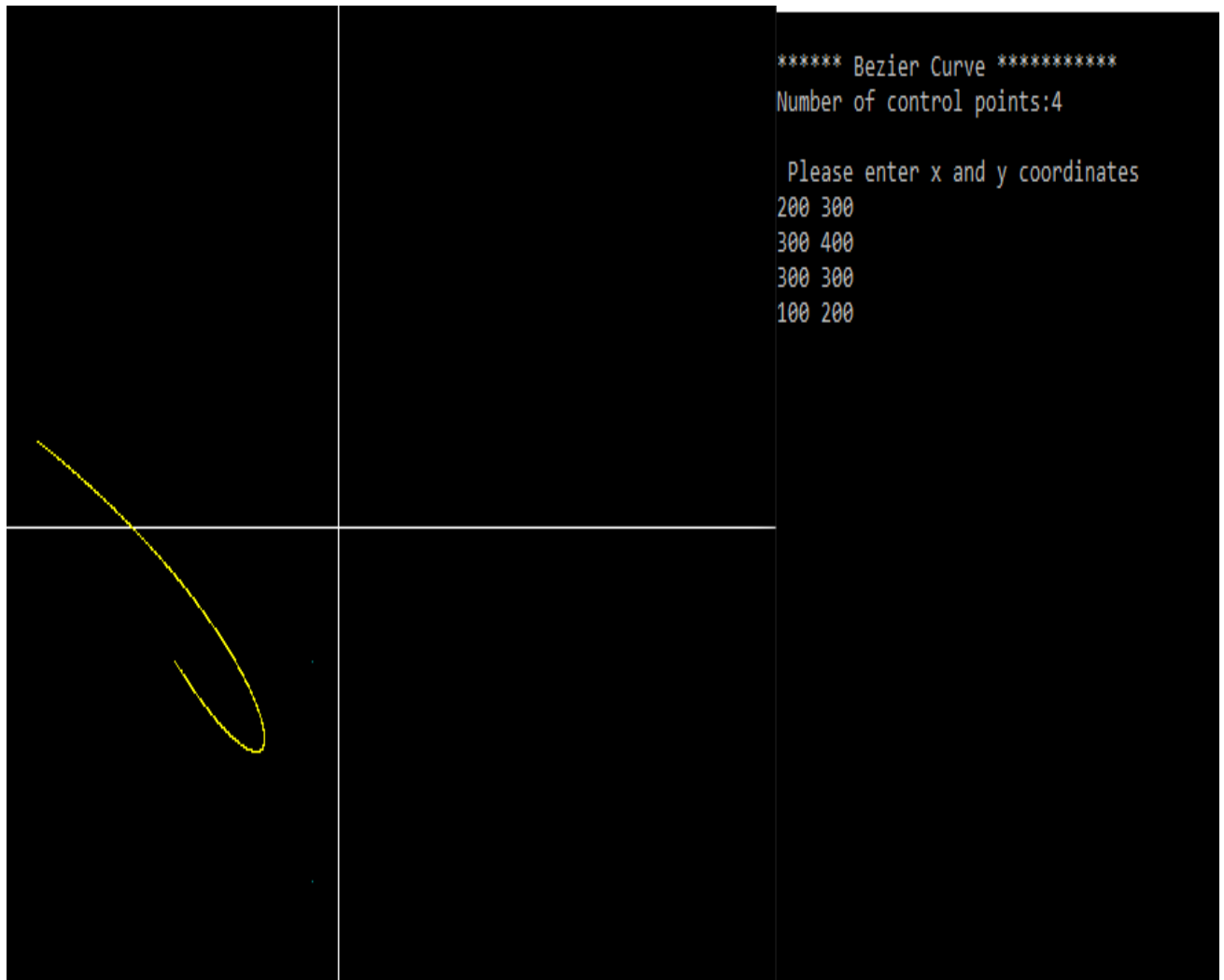
getch();

closegraph();

}

```

## OUTPUT:



## **PROGRAM – 13(Extra work)**

**PROBLEM STATEMENT:** Write a program to implement Hyperbola

### **CODE:**

```
#include<graphics.h>
#include<conio.h>
#include<stdio.h>
#include<dos.h>
#include<stdlib.h>
int main()
{
int gd=DETECT;

int gm;

initgraph(&gd,&gm,NULL);

float x,y;

float xc,yc;

float d,fx,fy,b,a;

d=b*b*(a+0.5)*(a+0.5)-a*a-a*a*b*b;

printf("enter centre (xc,yc)\n");

scanf("%f %f",&xc,&yc);

printf("enter a & b");

scanf("%f %f",&a,&b);

x=a;
```

```
y=0;
```

```
fx=2*b*b*a;
```

```
fy=0;
```

```
setcolor(WHITE);
```

```
line(0,240,640,240);
```

```
line(320,0,320,480);
```

```
while(abs(fy)<=fx)
```

```
{
```

```
if(d>=0)
```

```
{
```

```
d=d-a*a*(2*y+3);
```

```
}
```

```
else
```

```
{
```

```
d=d-a*a*(2*y+3)+b*b*(2*x+2);
```

```
x++;
```

```
fx=fx+2*b*b;
```

```
}
```

```
y++;
```

```
fy=fy+2*a*a;
```

```
putpixel(x+320+xc,240-y-yc,YELLOW);
```

```
putpixel(x+320+xc,240+y-yc,YELLOW);
```

```
putpixel(-x+320+xc,240-y-yc,YELLOW);
```

```
delay(20);
```

```
putpixel(-x+320+xc,240+y-yc,YELLOW);
```

```
delay(20);
```

```
}
```

```
int p;
```

```
x=p/2;
```

```
y=p;
```

```
d=-p;
```

```
while(y<3*p)
```

```
{
```

```
x++;
```

```
if(d>=0)
```

```
{
```

```
d=d-2*p;
```



```
}
```

```
else
```

```
{
```

```
d=d+2*y+2-2*p;
```

```
y++;
```

```
}
```

```
putpixel(x+320+xc,240-y-yc,RED);
```

```
delay(20);
```

```
putpixel(x+320+xc,240+y-yc,RED);
```

```
delay(20);
```

```
}
```

```
getch();
```

```
}
```

**OUTPUT:**





**AMITY UNIVERSITY**  
UTTAR PRADESH

## **COMPUTER GRAPHICS [CSE203] OPEN ENDED**

**Submitted by:-**

**Name: Anchal kumari**

**Enrollment no./Sec: A12405218083(6cse3x)**

**Submitted To:-**

**Dr. Renuka Nagpal**



**AMITY SCHOOL OF ENGINEERING AND  
TECHNOLOGY AMITY UNIVERSITY CAMPUS,  
SECTOR-125, NOIDA-201301**

**Problem statement:** Write a program to implement tower of Hanoi game using DDA line drawing algorithm.

**Theory:**

DDA stands for Digital Differential Analyzer. It is an incremental method of scan conversion of line. In this method calculation is performed at each step but by using results of previous steps.

**Case1:** When  $|M| < 1$  then (assume that  $x_1 < x_2$ )

$x = x_1, y = y_1$  set  $\Delta x = 1$

$y_{i+1} = y_1 + m, \quad x = x + 1$

Until  $x = x_2$

**Case2:** When  $|M| > 1$  then (assume that  $y_1 < y_2$ )

$x = x_1, y = y_1$  set  $\Delta y = 1$

$x_{i+1} = x_1 + 1/m, \quad y = y + 1$

Until  $y \rightarrow y_2$

**Advantage:**

1. It is a faster method than method of using direct use of line equation.
2. This method does not use multiplication theorem.
3. It allows us to detect the change in the value of  $x$  and  $y$ , so plotting of same point twice is not possible.
4. This method gives overflow indication when a point is repositioned.
5. It is an easy method because each step involves just two additions.

**Disadvantage:**

1. It involves floating point additions rounding off is done. Accumulations of round off error cause accumulation of error.
2. Rounding off operations and floating point operations consumes a lot of time.
3. It is more suitable for generating line using the software. But it is less suited for hardware implementation.

## **CODE:**

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

int tower[3][10]; // the three towers' disks as stack
int top[3]; // top of the three stacks
int from,to; // moving 'from' tower number 'to' tower number
int diskInAir; // number of disk moved (1 to n)
int l,b,u;

//dda function//
void dda(float x1,float y1,float x2,float y2){
float x,y,dx,dy,step,i;
dx=abs(x2-x1);
dy=abs(y2-y1);
if(dx>=dy)
step=dx;
else
step=dy;
dx=dx/step;
dy=dy/step;
x=x1;
y=y1;
i=1;
while(i<=step)
{
putpixel(x,y,15);
x=x+dx;
y=y+dy;
i=i+1;
}
}
//dda function

void push(int to, int diskno)
//putting disk on tower
{
tower[to-1][++top[to-1]]=diskno;
}

int pop(int from)
//take topmost disk from tower
{
return(tower[from-1][top[from-1]--]);
}

void drawStill()
```

```

{
int j,i,disk;
cleardevice();
for(j=1;j<=3;j++)
{
//draw tower
setfillstyle(CLOSE_DOT_FILL,WHITE);
dda(j*1,u,j*1,b);
dda(j*1+2,u,j*1+2,b);
dda(j*1,u,j*1,b);
// bar(j*1,u,j*1+5,b);
//draw all disks on tower
for(i=0;i<=top[j-1];i++)
{
disk=tower[j-1][i];
setfillstyle(SOLID_FILL,1+disk);

bar(j*1-15-disk*5,b-(i+1)*10,j*1+5+15+disk*5,b-i*10);
}
}
}

```

```

void animator()
//to show the movement of disk
{
int x,y,dif,sign;
diskInAir=pop(from);
x=from*1;
y=b-(top[from-1]+1)*10;
//taking disk upward from the tower
for(;y>u-20;y-=15)
{
drawStill();
setfillstyle(SOLID_FILL,1+diskInAir);
bar(x-15-diskInAir*5,y-10,x+5+15+diskInAir*5,y);
delay(100);
}
y=u-20;
dif=to*1-x;
sign=dif/abs(dif);
//moving disk towards a target tower
for(;abs(x-to*1)>25;x+=sign*15)
{
drawStill();
setfillstyle(SOLID_FILL,1+diskInAir);
bar(x-15-diskInAir*5,y-10,x+5+15+diskInAir*5,y);
delay(100);
}
x=to*1;
//placing disk on a target tower

```

```

for(;y<b-(top[to-1]+1)*10;y+=15)
{
drawStill();
setfillstyle(SOLID_FILL,1+diskInAir);
bar(x-15-diskInAir*5,y-10,x+5+15+diskInAir*5,y);
delay(100);
}
push(to,diskInAir);
drawStill();
}

```

```

void moveTopN(int n, int a, int b, int c)
//Move top n disk from tower 'a' to tower 'c'
//tower 'b' used for swapping
{
if(n>=1)
{
moveTopN(n-1,a,c,b);
drawStill();
delay(1000);
from=a;
to=c;
//animating the move
animator();
moveTopN(n-1,b,a,c);
}
}

```

```

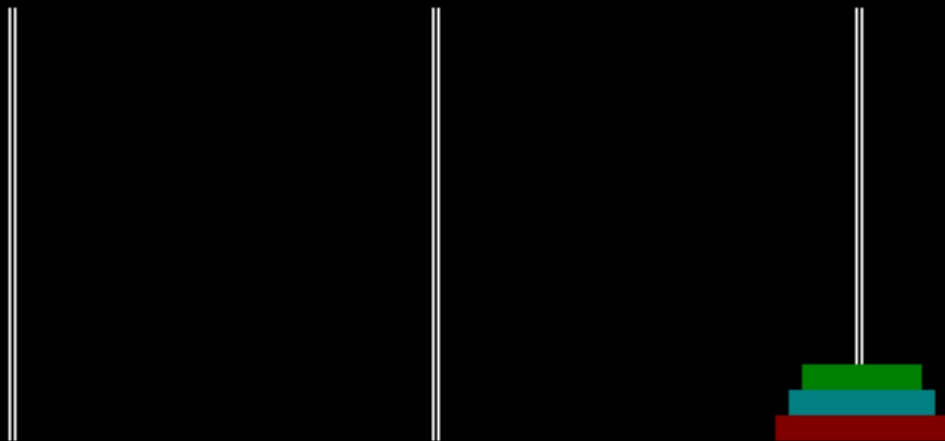
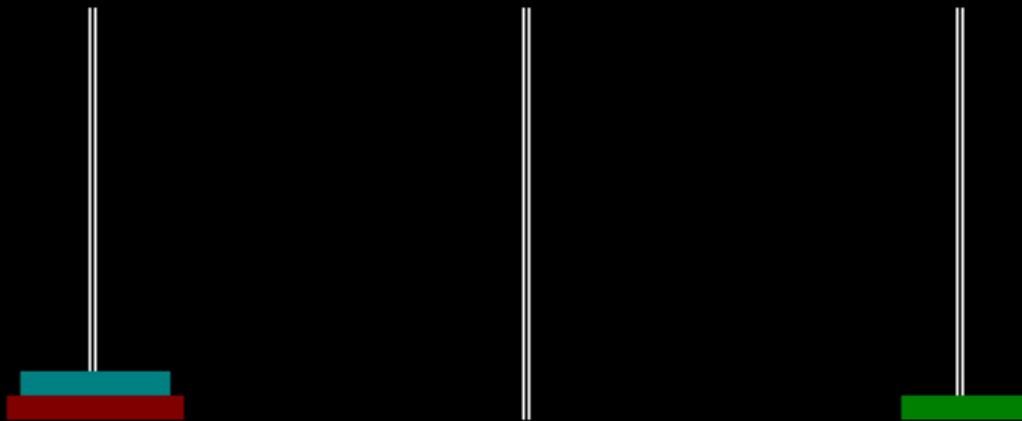
int main()
{
int i,gd=DETECT,gm,n;
printf("Enter number of disks");
scanf("%d",&n);
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI\\");
//setting all tower empty
for(i=0;i<3;i++)
{
top[i]=-1;
}
//putting all disks on tower 'a'
for(i=n;i>0;i--)
{
push(1,i);
}
l=getmaxx()/4;
b=getmaxy()-50;
u=getmaxy()/3+100;
//start solving
moveTopN(n,1,2,3);
delay(4000);
}

```

```
getch();  
}
```

### **OUTPUT:**

Enter number of disks3





**Problem statement:** Write a program to implement moving wheel using DDA line drawing and mid-point circle algorithm.

### **Theory:**

DDA stands for Digital Differential Analyzer. It is an incremental method of scan conversion of line. In this method calculation is performed at each step but by using results of previous steps.

**Case1:** When  $|M| < 1$  then (assume that  $x_1 < x_2$ )

$x = x_1, y = y_1$  set  $\Delta x = 1$

$y_{i+1} = y_1 + m, \quad x = x + 1$

Until  $x = x_2$

**Case2:** When  $|M| < 1$  then (assume that  $y_1 < y_2$ )

$x = x_1, y = y_1$  set  $\Delta y = 1$

$x_{i+1} = x_1 + 1/m, \quad y = y + 1$

Until  $y = y_2$

### **Advantage:**

6. It is a faster method than method of using direct use of line equation.
7. This method does not use multiplication theorem.
8. It allows us to detect the change in the value of  $x$  and  $y$ , so plotting of same point twice is not possible.
9. This method gives overflow indication when a point is repositioned.
10. It is an easy method because each step involves just two additions.

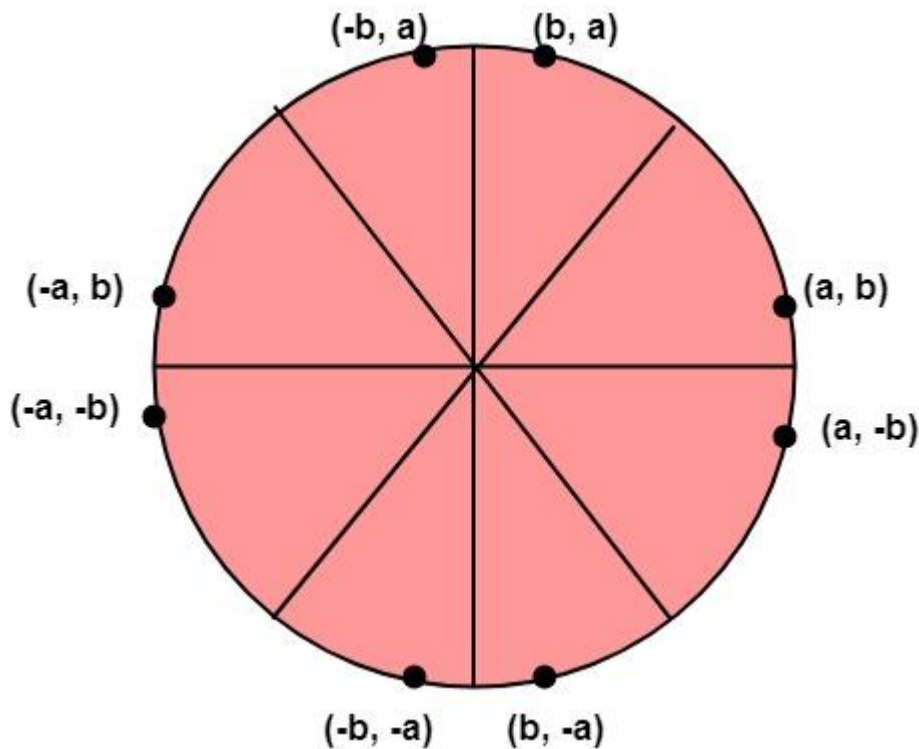
### **Disadvantage:**

4. It involves floating point additions rounding off is done. Accumulations of round off error cause accumulation of error.
5. Rounding off operations and floating point operations consumes a lot of time.
6. It is more suitable for generating line using the software. But it is less suited for hardware implementation.

## MidPoint Circle Algorithm

It is based on the following function for testing the spatial relationship between the arbitrary point (x, y) and a circle of radius r centered at the origin:

$$f(x, y) = x^2 + y^2 - r^2 \quad \left[ \begin{array}{l} < 0 \text{ for } (x, y) \text{ inside the circle} \\ = 0 \text{ for } (x, y) \text{ on the circle} \\ > 0 \text{ for } (x, y) \text{ outside the circle} \end{array} \right] \dots \text{equation 1}$$



Now, consider the coordinates of the point halfway between pixel T and pixel S

This is called midpoint  $(x_{i+1}, y_i - \frac{1}{2})$  and we use it to define a decision parameter:

$$P_i = f(x_{i+1}, y_i - \frac{1}{2}) = (x_{i+1})^2 + (y_i - \frac{1}{2})^2 - r^2 \dots \text{equation 2}$$

If  $P_i$  is -ve  $\Rightarrow$  midpoint is inside the circle and we choose pixel T

If  $P_i$  is +ve  $\Rightarrow$  midpoint is outside the circle (or on the circle) and we choose pixel S.

The decision parameter for the next step is:

$$P_{i+1} = (x_{i+1} + 1)^2 + (y_{i+1} - \frac{1}{2})^2 - r^2 \dots \text{equation 3}$$

Since  $x_{i+1} = x_{i+1}$ , we have

$$\begin{aligned}
P_{i+1} - P_i &= ((x_i + 1) + 1)^2 - (x_i + 1)^2 + (y_{i+1} - \frac{1}{2})^2 - (y_i - \frac{1}{2})^2 \\
&= x_i^2 + 4 + 4x_i - x_i^2 + 1 - 2x_i + y_{i+1}^2 + \frac{1}{4} - y_{i+1} - y_i^2 - \frac{1}{4} - y_i \\
&= 2(x_i + 1) + 1 + (y_{i+1}^2 - y_i^2) - (y_{i+1} - y_i) \dots \dots \dots \text{equation 4} \\
P_{i+1} &= P_i + 2(x_i + 1) + 1 + (y_{i+1}^2 - y_i^2) - (y_{i+1} - y_i) \dots \dots \dots \text{equation 4}
\end{aligned}$$

If pixel T is chosen  $\Rightarrow P_i < 0$

We have  $y_{i+1} = y_i$

If pixel S is chosen  $\Rightarrow P_i \geq 0$

We have  $y_{i+1} = y_i - 1$

Thus, 
$$P_{i+1} = \begin{cases} P_i + 2(x_i + 1) + 1, & \text{if } P_i < 0 \\ P_i + 2(x_i + 1) + 1 - 2(y_i - 1), & \text{if } P_i \geq 0 \end{cases} \dots \dots \dots \text{equation 5}$$

We can continue to simplify this in n terms of  $(x_i, y_i)$  and get

$$P_{i+1} = \begin{cases} P_i + 2x_i + 3, & \text{if } P_i < 0 \\ P_i + 2(x_i - y_i) + 5, & \text{if } P_i \geq 0 \end{cases} \dots \dots \dots \text{equation 6}$$

Now, initial value of  $P_i$  (0,r) from equation 2

$$\begin{aligned}
P_1 &= (0 + 1)^2 + (r - \frac{1}{2})^2 - r^2 \\
&= 1 + \frac{1}{4} - r^2 = \frac{5}{4} - r
\end{aligned}$$

We can put  $\frac{5}{4} \cong 1$   
 $\therefore r$  is an integer  
 So,  $P_1 = 1 - r$

## **CODE:**

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

int l = 1;

void ddaline(int x1, int y1, int x2, int y2) {
    int s, dx, dy, m;
    float xi, yi, x, y;
    dx = x2 - x1;
    dy = y2 - y1;
    if (abs(dx) > abs(dy))
        s = abs(dx); else
        s = abs(dy);
    xi = dx / (float) s;
    yi = dy / (float) s;
    x = x1;
    y = y1;
    putpixel(x1 + 0.5, y1 + 0.5, 15);
    for (m = 0; m < s; m++) {
        x += xi;
        y += yi;
        putpixel(x + 0.5, y + 0.5, 15);
    }
}

void plotpoints1(int x, int y, int cx, int cy) {
    putpixel(cx + x, cy + y, 15);
    putpixel(cx - x, cy - y, 15);
    putpixel(cx - y, cy + x, 15);
```

```

        putpixel(cx + y, cy - x, 15);
        if (l % 20 == 0) {
            ddaline(cx - x, cy - y, cx + x, cy + y);
            ddaline(cx - y, cy + x, cx + y, cy - x);
        }
        l++;
    }

void plotpoints2(int x, int y, int cx, int cy) {
    putpixel(cx - x, cy + y, 14);
    putpixel(cx + x, cy - y, 14);
    putpixel(cx + y, cy + x, 14);
    putpixel(cx - y, cy - x, 14);
    if (l % 20 == 0) {
        ddaline(cx + x, cy - y, cx - x, cy + y);
        ddaline(cx - y, cy - x, cx + y, cy + x);
    }
    l++;
}

void mcircle(int cx, int cy, int r) {
    int x = 0, y, p;
    y = r;
    p = 1 - r;
    while (x < y) {
        plotpoints1(x, y, cx, cy);
        x++;
        if (p < 0)
            p += 2 * x + 1; else {
                y--;
                p += 2 * (x - y) + 1;
            }
    }
}

```

```

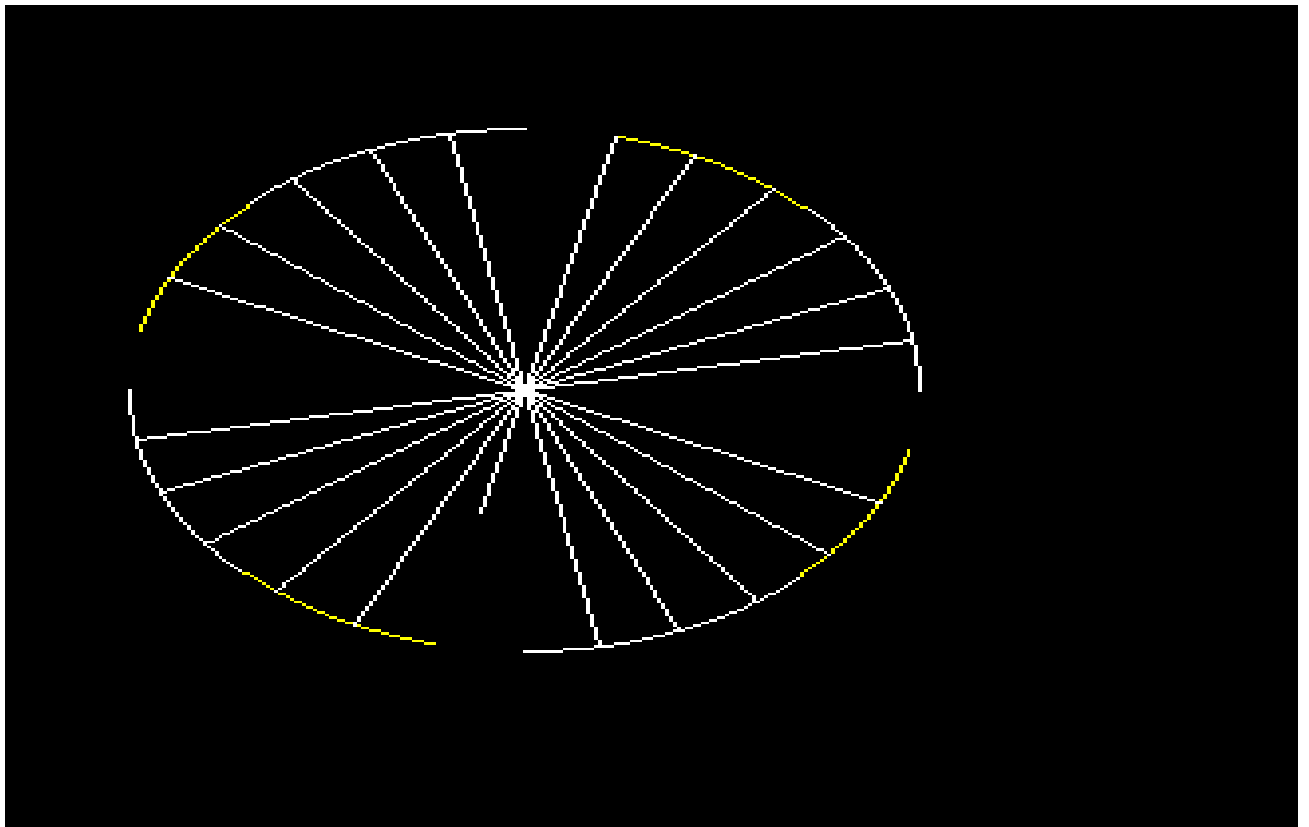
    }
    x = y + 1;
    while (abs(x) > 0) {
        plotpoints2(x, y, cx, cy);
        x--;
        if (p >= 0)
            p = p - 2 * x - 1; else {
                y++;
                p = p - 2 * (x - y) - 1;
            }
    }
}

int main() {
    int gd = DETECT, gm = DETECT;
    int i = 0;
    initgraph(&gd, &gm, "");
    while (!kbhit()) {
        if (i > 640)
            i = -200;

        cleardevice();
        mcircle(100 + (i++), 200, 100);
        delay(10);
        i++;
    }
    getch();
}

```

**OUTPUT:**



**Problem statement:** Write a program to implement tic tac toe game using DDA line drawing and mid-point circle algorithm.

**CODE:**

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

//dda function//
void dda(int x1,int y1,int x2,int y2){
float x,y,dx,dy,step,i;
dx=abs(x2-x1);
dy=abs(y2-y1);
if(dx>=dy)
step=dx;
else
step=dy;
dx=dx/step;
dy=dy/step;
x=x1;
y=y1;
i=1;
while(i<=step)
{
putpixel(x,y,10);
x=x+dx;
y=y+dy;
```



```

i=i+1;
}
}
//dda function
void midpointcircle(int x0, int y0, int radius)
{
    int x = radius;
    int y = 0;
    int err = 0;

    while (x >= y)
    {
        putpixel(x0 + x, y0 + y, 4);
        putpixel(x0 + y, y0 + x, 4);
        putpixel(x0 - y, y0 + x, 4);
        putpixel(x0 - x, y0 + y, 4);
        putpixel(x0 - x, y0 - y, 4);
        putpixel(x0 - y, y0 - x, 4);
        putpixel(x0 + y, y0 - x, 4);
        putpixel(x0 + x, y0 - y, 4);

        if (err <= 0)
        {
            y += 1;
            err += 2*y + 1;
        }

        if (err > 0)
        {
            x -= 1;

```

```

        err -= 2*x + 1;

    }

}

}

//Declaration Part Starts

    bool B1=true,B2=true,B3=true,B4=true,B5=true,B6=true,B7=true,B8=true,B9=true; //B--
-Block

    char turn='X';

    int turn_count=0;

    POINT CP;

    char Cursor_Readings[100];

    int
B1_W=NULL,B2_W=NULL,B3_W=NULL,B4_W=NULL,B5_W=NULL,B6_W=NULL,B
7_W=NULL,B8_W=NULL,B9_W=NULL; //B--Block, W-Winner

    int Winner=0;

//Declaration Part ends

void Check_For_A_Winner()

{

    if(B1_W==1&&B2_W==1&&B3_W==1)

    {

        outtextxy(10,60,"Game Over...");

        B1=B2=B3=B4=B5=B6=B7=B8=B9=false;

        Winner=1;

    }

    if(B4_W==1&&B5_W==1&&B6_W==1)

    {

        outtextxy(10,60,"Game Over...");

        B1=B2=B3=B4=B5=B6=B7=B8=B9=false;

        Winner=1;

    }

}

```

```
if(B7_W==1&&B8_W==1&&B9_W==1)
{
    outtextxy(10,60,"Game Over...");
    B1=B2=B3=B4=B5=B6=B7=B8=B9=false;
    Winner=1;
}
if(B1_W==1&&B5_W==1&&B9_W==1)
{
    outtextxy(10,60,"Game Over...");
    B1=B2=B3=B4=B5=B6=B7=B8=B9=false;
    Winner=1;
}
if(B3_W==1&&B5_W==1&&B7_W==1)
{
    outtextxy(10,60,"Game Over...");
    B1=B2=B3=B4=B5=B6=B7=B8=B9=false;
    Winner=1;
}
if(B1_W==1&&B4_W==1&&B7_W==1)
{
    outtextxy(10,60,"Game Over...");
    B1=B2=B3=B4=B5=B6=B7=B8=B9=false;
    Winner=1;
}
if(B2_W==1&&B5_W==1&&B8_W==1)
{
    outtextxy(10,60,"Game Over...");
    B1=B2=B3=B4=B5=B6=B7=B8=B9=false;
    Winner=1;
}
```

```
if(B3_W==1&&B6_W==1&&B9_W==1)
{
    outtextxy(10,60,"Game Over...");
    B1=B2=B3=B4=B5=B6=B7=B8=B9=false;
    Winner=1;
}
```

```
if(B1_W==2&&B2_W==2&&B3_W==2)
{
    outtextxy(10,60,"Game Over...");
    B1=B2=B3=B4=B5=B6=B7=B8=B9=false;
    Winner=2;
}
```

```
if(B4_W==2&&B5_W==2&&B6_W==2)
{
    outtextxy(10,60,"Game Over...");
    B1=B2=B3=B4=B5=B6=B7=B8=B9=false;
    Winner=2;
}
```

```
if(B7_W==2&&B8_W==2&&B9_W==2)
{
    outtextxy(10,60,"Game Over...");
    B1=B2=B3=B4=B5=B6=B7=B8=B9=false;
    Winner=2;
}
```

```
if(B1_W==2&&B5_W==2&&B9_W==2)
{
    outtextxy(10,60,"Game Over...");
    B1=B2=B3=B4=B5=B6=B7=B8=B9=false;
    Winner=2;
}
```

```

    }

    if(B3_W==2&&B5_W==2&&B7_W==2)
    {
        outtextxy(10,60,"Game Over...");
        B1=B2=B3=B4=B5=B6=B7=B8=B9=false;
        Winner=2;
    }

    if(B1_W==2&&B4_W==2&&B7_W==2)
    {
        outtextxy(10,60,"Game Over...");
        B1=B2=B3=B4=B5=B6=B7=B8=B9=false;
        Winner=2;
    }

    if(B2_W==2&&B5_W==2&&B8_W==2)
    {
        outtextxy(10,60,"Game Over...");
        B1=B2=B3=B4=B5=B6=B7=B8=B9=false;
        Winner=2;
    }

    if(B3_W==2&&B6_W==2&&B9_W==2)
    {
        outtextxy(10,60,"Game Over...");
        B1=B2=B3=B4=B5=B6=B7=B8=B9=false;Winner=2;
    }
}

void Turn_MessageX()
{
    settextstyle(1,0,1);
    if(turn_count==8)
        outtextxy(10,60,"Game Over...");
}

```

```

    else
        outtextxy(10,60,"O's Turn");
}
void Turn_MessageO()
{
    settextstyle(1,0,1);
    if(turn_count==8)
        outtextxy(10,60,"Game Over...");
    else
        outtextxy(10,60,"X's Turn");
}
void Reset_All()
{
    B1=B2=B3=B4=B5=B6=B7=B8=B9=true;
    turn='X';
    turn_count=0;
    B1_W=B2_W=B3_W=B4_W=B5_W=B6_W=B7_W=B8_W=B9_W=NULL;
    Winner=0;
}
void Allow_Access()
{
    B1=B2=B3=B4=B5=B6=B7=B8=B9=true;
}
void Deny_Access()
{
    B1=B2=B3=B4=B5=B6=B7=B8=B9=false;
}
void OOO(int x,int y)
{
    setcolor(20);

```

```

    // for(int a=0;a<=2;a++)
        midpointcircle(x,y,25);
        //circle(x,y,25+a);
    }
void cross(int x,int y)
{
    setcolor(10);
    // for(int a=0;a<=2;a++)
        {

dda((x-20),(y-20),(x+20),(y+20));
    //  dda((x+20),(y-20),(x-20),(y+20));

//line((x-20),(y-20),(x+20),(y+20));
    line((x+20),(y-20),(x-20),(y+20));


// line((x-20)-a,(y-20)+a,(x+20)-a,(y+20)+a);
//  line((x+20),(y-20),(x-20),(y+20));
    }
}

void GetStatus(int tempCx,int tempCy)
{
    settextstyle(1,0,1);
    if((tempCx>420&&tempCx<620&&tempCy>280&&tempCy<320))
    {
        outtextxy(460,410,"Exit      ");
    }
    else if((tempCx>420&&tempCx<620&&tempCy>180&&tempCy<220))

```

```
{
    outtextxy(460,410,"New Game    ");
}
else if(tempCx>110&&tempCx<210&&tempCy>130&&tempCy<230)
{
    if(B1)
        outtextxy(460,410,"Block-1    ");
    else
        outtextxy(460,410,"Not Allowed  ");
}
else if(tempCx>210&&tempCx<310&&tempCy>130&&tempCy<230)
{
    if(B2)
        outtextxy(460,410,"Block-2    ");
    else
        outtextxy(460,410,"Not Allowed  ");
}
else if(tempCx>310&&tempCx<410&&tempCy>130&&tempCy<230)
{
    if(B3)
        outtextxy(460,410,"Block-3    ");
    else
        outtextxy(460,410,"Not Allowed  ");
}
else if(tempCx>110&&tempCx<210&&tempCy>230&&tempCy<330)
{
    if(B4)
        outtextxy(460,410,"Block-4    ");
    else
        outtextxy(460,410,"Not Allowed  ");
}
```



```

}
else if(tempCx>210&&tempCx<310&&tempCy>230&&tempCy<330)
{
    if(B5)
        outtextxy(460,410,"Block-5      ");
    else
        outtextxy(460,410,"Not Allowed  ");
}
else if(tempCx>310&&tempCx<410&&tempCy>230&&tempCy<330)
{
    if(B6)
        outtextxy(460,410,"Block-6      ");
    else
        outtextxy(460,410,"Not Allowed  ");
}
else if(tempCx>110&&tempCx<210&&tempCy>330&&tempCy<430)
{
    if(B7)
        outtextxy(460,410,"Block-7      ");
    else
        outtextxy(460,410,"Not Allowed  ");
}
else if(tempCx>210&&tempCx<310&&tempCy>330&&tempCy<430)
{
    if(B8)
        outtextxy(460,410,"Block-8      ");
    else
        outtextxy(460,410,"Not Allowed  ");
}
else if(tempCx>310&&tempCx<410&&tempCy>330&&tempCy<430)

```

```

    {
        if(B9)
            outtextxy(460,410,"Block-9    ");
        else
            outtextxy(460,410,"Not Allowed    ");
    }
    else if(Winner==1)
    {
        outtextxy(460,410,"X Wins    ");
    }
    else if(Winner==2)
    {
        outtextxy(460,410,"O Wins    ");
    }
    else if(turn_count==9)
    {
        outtextxy(460,410,"Draw    ");
    }
    else
    {
        outtextxy(460,410,"Ready    ");
    }
}

void get_turn()
{
    if(turn_count==0)
        turn='X';
    if(turn_count==1)
        turn='O';
    if(turn_count==2)

```

```

        turn='X';
    if(turn_count==3)
        turn='O';
    if(turn_count==4)
        turn='X';
    if(turn_count==5)
        turn='O';
    if(turn_count==6)
        turn='X';
    if(turn_count==7)
        turn='O';
    if(turn_count==8)
        turn='X';
    if(turn_count==9)
        turn='O';
}

void NewGame()
{
    cleardevice();
    Reset_All();
    settextstyle(1,0,1);
    outtextxy(((getmaxx()/2)-textwidth("Loading")/2),((getmaxy()/2)-
textheight("Loading")/2),"Loading");
    for(int i=0;i<80;i++)
    {
        outtext(".");
        delay(10);
    }
    cleardevice();
}

void Boards_Titles()

```

```

{
    setcolor(15);

    settextstyle(1,0,1);

    outtextxy(10,410,Cursor_Readings);

    setcolor(3);

    settextstyle(1,0,5);

    outtextxy(150,10,"Tic-Tac-Toe");

    setcolor(10);

    rectangle(100,100,400,400);    //Main Board Border
    rectangle(420,150,620,200);    //New Game Board Border
    rectangle(420,250,620,300);    //Exit Board Border

    dda(200,100,200,400);

    dda(300,100,300,400);    //CL2
    dda(100,200,400,200);    //RL1
    dda(100,300,400,300);


    //line(200,100,200,400);    //CL1
    // line(300,100,300,400);    //CL2
    // line(100,200,400,200);    //RL1
    // line(100,300,400,300);    //RL2

    setcolor(15);

    settextstyle(1,0,1);

    outtextxy(450,165,"New Game");

    outtextxy(490,265,"Exit");

}

int main()
{

```

```

int gd=DETECT,gm;
initgraph(&gd,&gm,"");
settextstyle(1,0,1);
outtextxy(10,60,"X's Turn");
while(1)
{
    GetCursorPos(&CP);
    int Cx=CP.x,Cy=CP.y;
    //  sprintf(Cursor_Readings,"X : %d Y : %d",Cx,Cy);
    Boards_Titles();
    if(GetAsyncKeyState(VK_LBUTTON))
    {
        get_turn();
        if((Cx>420&&Cx<620&&Cy>280&&Cy<320))    //Exit
        {
            outtextxy(460,410,"Exiting.....  ");
            delay(500);
            outtextxy(460,410,"Done.      ");
            exit(0);
        }
        else if(Cx>420&&Cx<620&&Cy>180&&Cy<280)    //New Game
        {
            NewGame();
            settextstyle(1,0,1);
            outtextxy(10,60,"X's Turn");
        }
        else if(Cx>110&&Cx<210&&Cy>130&&Cy<230&&B1)    //Block-1
        {
            if(turn=='X')
            {

```

```

        Turn_MessageX();
        cross(150,150);
        B1_W=1;
    }
    if(turn=='O')
    {
        Turn_MessageO();
        OOO(150,150);
        B1_W=2;
    }
    B1=false;
    turn_count++;
}
else if(Cx>210&&Cx<310&&Cy>130&&Cy<230&&B2)    //Block-2
{
    if(turn=='X')
    {
        Turn_MessageX();
        cross(250,150);
        B2_W=1;
    }
    if(turn=='O')
    {
        Turn_MessageO();
        OOO(250,150);
        B2_W=2;
    }
    B2=false;
    turn_count++;
}

```

```

}
else if(Cx>310&&Cx<410&&Cy>130&&Cy<230&&B3)    //Block-3
{
    if(turn=='X')
    {
        Turn_MessageX();
        cross(350,150);
        B3_W=1;
    }
    if(turn=='O')
    {
        Turn_MessageO();
        OOO(350,150);
        B3_W=2;
    }
    B3=false;
    turn_count++;

}

else if(Cx>110&&Cx<210&&Cy>230&&Cy<330&&B4)    //Block-4
{
    if(turn=='X')
    {
        Turn_MessageX();
        cross(150,250);
        B4_W=1;
    }
    if(turn=='O')
    {
        Turn_MessageO();

```

```

        OOO(150,250);

        B4_W=2;

    }

    B4=false;

    turn_count++;

}

else if(Cx>210&&Cx<310&&Cy>230&&Cy<330&&B5)    //Block-5
{

    if(turn=='X')

    {

        Turn_MessageX();

        cross(250,250);

        B5_W=1;

    }

    if(turn=='O')

    {

        Turn_MessageO();

        OOO(250,250);

        B5_W=2;

    }

    B5=false;

    turn_count++;

}

else if(Cx>310&&Cx<410&&Cy>230&&Cy<330&&B6)    //Block-6
{

    if(turn=='X')

    {

        Turn_MessageX();

        cross(350,250);

        B6_W=1;

    }

    if(turn=='O')

    {

        Turn_MessageO();

        cross(250,350);

        B6_W=2;

    }

    B6=false;

    turn_count++;

}

}

```



```

    }
    if(turn=='O')
    {
        Turn_MessageO();
        OOO(350,250);
        B6_W=2;
    }
    B6=false;
    turn_count++;

}

else if(Cx>110&&Cx<210&&Cy>330&&Cy<430&&B7)    //Block-7
{
    if(turn=='X')
    {
        Turn_MessageX();
        cross(150,350);
        B7_W=1;
    }
    if(turn=='O')
    {
        Turn_MessageO();
        OOO(150,350);
        B7_W=2;
    }
    B7=false;
    turn_count++;
}

else if(Cx>210&&Cx<310&&Cy>330&&Cy<430&&B8)    //Block-8
{

```

```

    if(turn=='X')
    {
        Turn_MessageX();
        cross(250,350);
        B8_W=1;
    }
    if(turn=='O')
    {
        Turn_MessageO();
        OOO(250,350);
        B8_W=2;
    }
    B8=false;
    turn_count++;
}
else if(Cx>310&&Cx<410&&Cy>330&&Cy<430&&B9)    //Block-9
{
    if(turn=='X')
    {
        Turn_MessageX();
        cross(350,350);
        B9_W=1;
    }
    if(turn=='O')
    {
        Turn_MessageO();
        OOO(350,350);
        B9_W=2;
    }
    B9=false;

```

```

        turn_count++;
    }
    Check_For_A_Winner();
}
GetStatus(Cx,Cy);
delay(10);
}
getch();
closegraph();
return 0;
}

```

### OUTPUT:



**Problem statement:** Write a program to implement solar system using graphics.

**CODE:**

```
#include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <dos.h>

#include <math.h>


/* manipulates the position of planets on the orbit */

void planetMotion(int xrad, int yrad, int midx, int midy, int x[60], int y[60]) {
    int i, j = 0;


    /* positions of planets in their corresponding orbits */
    for (i = 360; i > 0; i = i - 6) {
        x[j] = midx - (xrad * cos((i * 3.14) / 180));
        y[j++] = midy - (yrad * sin((i * 3.14) / 180));
    }
    return;
}


int main() {
    /* request auto detection */
    int gdriver = DETECT, gmode, err;
    int i = 0, midx, midy;
    int xrad[9], yrad[9], x[9][60], y[9][60];
    int pos[9], planet[9], tmp;
```

```

/* initialize graphic mode */
initgraph(&gdriver, &gmode, "C:/TURBOC3/BGI");
err = graphresult();

if (err != grOk) {
    /* error occurred */
    printf("Graphics Error: %s",
           grapherrormsg(err));
    return 0;
}

/* mid positions at x and y-axis */
midx = getmaxx() / 2;
midy = getmaxy() / 2;

/* manipulating radius of all 9 planets */
planet[0] = 7;
for (i = 1; i < 9; i++) {
    planet[i] = planet[i - 1] + 1;
}

/* offset position for the planets on their corresponding orbit */
for (i = 0; i < 9; i++) {
    pos[i] = i * 6;
}

/* orbits for all 9 planets */
xrad[0] = 60, yrad[0] = 30;
for (i = 1; i < 9; i++) {

```

```

        xrad[i] = xrad[i - 1] + 30;
        yrad[i] = yrad[i - 1] + 15;
    }

    /* positions of planets on their corresponding orbits */
    for (i = 0; i < 9; i++) {
        planetMotion(xrad[i], yrad[i], midx, midy, x[i], y[i]);
    }

    while (!kbhit()) {
        /* drawing 9 orbits */
        setcolor(WHITE);
        for (i = 0; i < 9; i++) {
            ellipse(midx, midy, 0, 360, xrad[i], yrad[i]);
        }

        /* sun at the mid of the solar system */
        setcolor(YELLOW);
        setfillstyle(SOLID_FILL, YELLOW);
        circle(midx, midy, 20);
        floodfill(midx, midy, YELLOW);

        /* mercury in first orbit */
        setcolor(CYAN);
        setfillstyle(SOLID_FILL, CYAN);
        pieslice(x[0][pos[0]], y[0][pos[0]], 0, 360, planet[0]);

        /* venus in second orbit */
        setcolor(GREEN);

```

```
setfillstyle(SOLID_FILL, GREEN);  
pieslice(x[1][pos[1]], y[1][pos[1]], 0, 360, planet[1]);
```

```
/* earth in third orbit */  
setcolor(BLUE);  
setfillstyle(SOLID_FILL, BLUE);  
pieslice(x[2][pos[2]], y[2][pos[2]], 0, 360, planet[2]);
```

```
/* mars in fourth orbit */  
setcolor(RED);  
setfillstyle(SOLID_FILL, RED);  
pieslice(x[3][pos[3]], y[3][pos[3]], 0, 360, planet[3]);
```

```
/* jupiter in fifth orbit */  
setcolor(BROWN);  
setfillstyle(SOLID_FILL, BROWN);  
pieslice(x[4][pos[4]], y[4][pos[4]], 0, 360, planet[4]);
```

```
/* saturn in sixth orbit */  
setcolor(LIGHTGRAY);  
setfillstyle(SOLID_FILL, LIGHTGRAY);  
pieslice(x[5][pos[5]], y[5][pos[5]], 0, 360, planet[5]);
```

```
/* uranus in sevth orbit */  
setcolor(BROWN);  
setfillstyle(SOLID_FILL, BROWN);  
pieslice(x[6][pos[6]], y[6][pos[6]], 0, 360, planet[6]);
```

```
/* neptune in eigth orbit */  
setcolor(LIGHTBLUE);
```

```

    setfillstyle(SOLID_FILL, LIGHTBLUE);
    pieslice(x[7][pos[7]], y[7][pos[7]], 0, 360, planet[7]);

    /* pluto in ninth orbit */
    setcolor(LIGHTRED);
    setfillstyle(SOLID_FILL, LIGHTRED);
    pieslice(x[8][pos[8]], y[8][pos[8]], 0, 360, planet[8]);

    /* checking for one complete rotation */
    for (i = 0; i < 9; i++) {
        if (pos[i] <= 0) {
            pos[i] = 59;
        } else {
            pos[i] = pos[i] - 1;
        }
    }

    /* sleep for 100 milliseconds */
    delay(100);

    /* clears graphic screen */
    cleardevice();
}

/* deallocate memory allocated for graphic screen */
closegraph();
return 0;
}

```



**OUTPUT:**

