

# **COMPUTER GRAPHICS LABORATORY**

## **LAB PRACTICALS RECORD**

**(CSX - 331)**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Dr. B R AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY**

**JALANDHAR – 144011, PUNJAB (INDIA)**

**July – December 2015**

**Submitted To:**

Ms Jagriti  
Assistant Professor ,  
Dept. Of CSE  
NIT Jalandhar

**Submitted by:**

Aman Garg  
13103050  
6<sup>th</sup> Semester  
Group G-3

**INDEX**

<b>S. No.</b>	<b>Title</b>	<b>Page</b>	<b>Sign</b>
1.	Program to implement various functions of graphics	3	
2.	Program to draw a line using direct method	5	
3.	Program to implement Digital Differential Analyzer	7	
4.	Program to draw a line using Bresenham's Line algorithm	11	
5.	Program to implement the polar co-ordinates method to draw circle	15	
6.	Program to implement mid point circle algorithm	17	
7.	Program to draw ellipse using Bresenham's algorithm	20	
8.	Program to perform 2D transformations : translation, rotation, scaling, shearing	24	
9.			
10.			
11.			
12.			

## 1. Write a program to implement various functions of graphics

### **Description:**

The first line to look at is: **graphics.h**. This file contains definitions and explanation of all the graphic functions and constants. In graphics, we have two modes: text mode and graphic mode. In text mode, it is possible to display or capture only text in terms of ASCII.

But in graphics mode, any type of figure can be displayed, captured and animated. To switch from text mode to graphic mode, we have a function called as "initgraph".

### **initgraph**

: This function initializes the graphic mode. It selects the best resolution and directs that value to mode in variable gm. The two int variables gd, gm are graphic driver and graphic mode respectively. The gm handles value that tells us which resolution and monitor we are using. The gd specifies the graphic driver to be used.

In our program we have gd=DETECT means we have passed the highest possible value available for the detected driver. If you don't want that value then you have to assign the constant value for gd and gm. The "&" symbol is used for initgraph to pass address of the constants.

### **closegraph()** :

The closegraph() switches back the screen from graphics mode to text mode. If you don't use this function then you may have undesirable effects.

### **Program:**

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

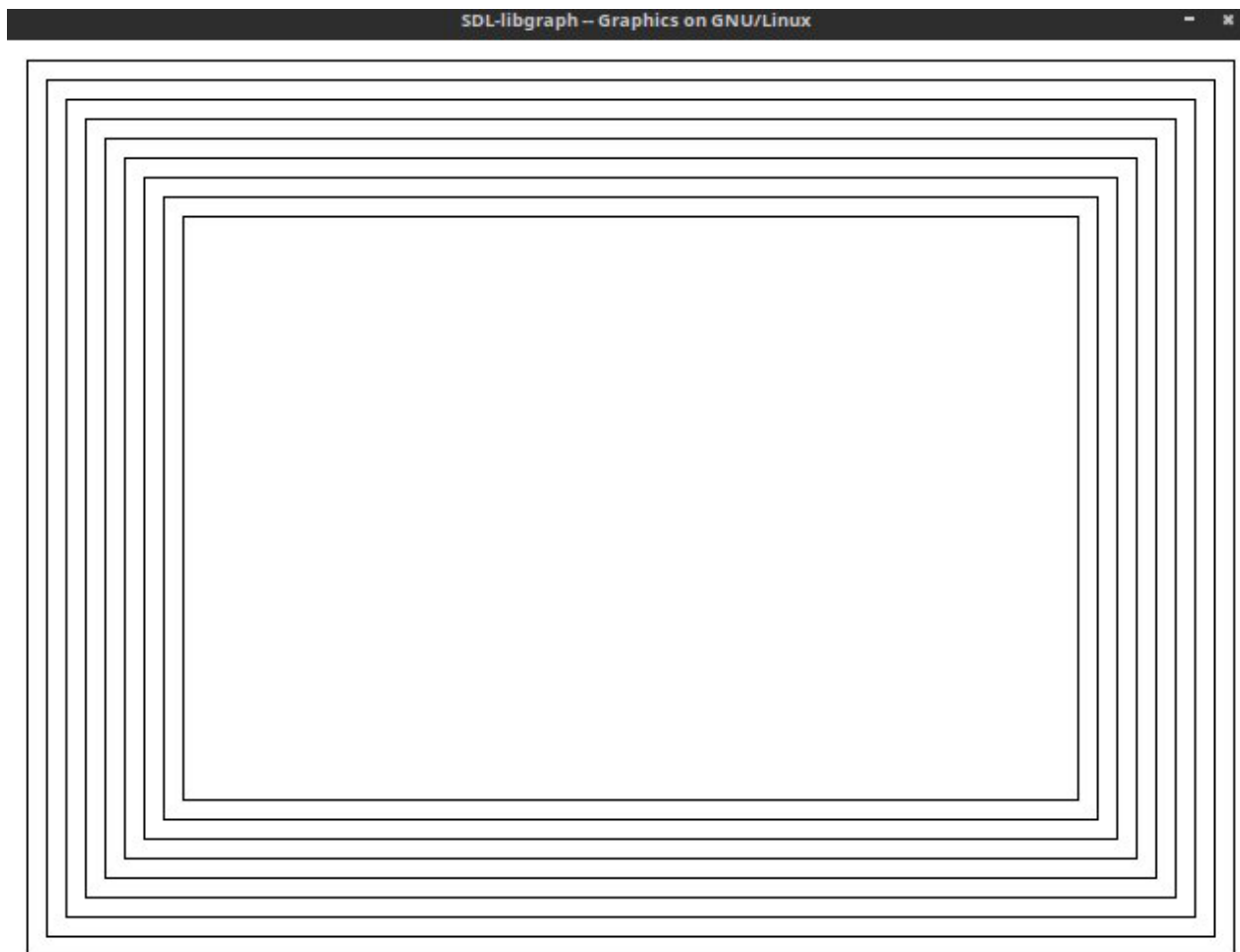
int main(){

    int graphDriver = DETECT, graphMode;
    initgraph(&graphDriver, &graphMode, NULL);
```

```
fprintf(stdout,"MAX : (%d, %d)\n",getmaxx(), getmaxy() );

// Creating a rectangular loop around the borders of concentric width 20
for (int i = 1; i < 10; i++)
    rectangle(10*i, getmaxy() - 10*i, getmaxx() - 10*i, 10*i);

delay(100000);
closegraph();
return 0;
}
```

**OUTPUT:**

<b>2.</b>	<b>Write a program to draw a line using Direct Method</b>
-----------	---

**Description:**

The equation of a straight line is given by:  $y=m.x+b$

Algorithm: Direct Method Algorithm

1. Start at the pixel for the left-hand endpoint  $x_1$
2. Step along the pixels horizontally until we reach the right-hand end of the line,  $x_r$
3. For each pixel compute the corresponding  $y$  value
4. Round this value to the nearest integer to select the nearest pixel

The algorithm performs a floating-point multiplication for every step in  $x$ . This method therefore requires an enormous number of floating-point multiplications, and is therefore expensive.

**Program:**

```
#include <graphics.h>
#include <stdio.h>
#include <assert.h>
#define coordinateFile "coordinates"
FILE * fp;

void directDrawLine(int x1, int y1, int x2, int y2){
    int dx = x2 - x1, dy = y2 - y1;
    float m = (dy * 1.)/dx; //slope
    float c = y1 - m*x1;
    int x = x1, y = y1;
    if (x2 < x1) x = x2, y = y2;

    for( int i = 0; i < abs(dx); i++){
        fprintf(stdout, "Plotting points (%d, %d) m: %.2f\n",x, y, m);
        putpixel(x, getmaxy() - y, BLACK);
        if (m <= 1.0){
            x = x + 1;
            y = m*x + c;
        }
        else {
```

```

        y = y + 1;
        x = (y - c)/m;
    }
}
}

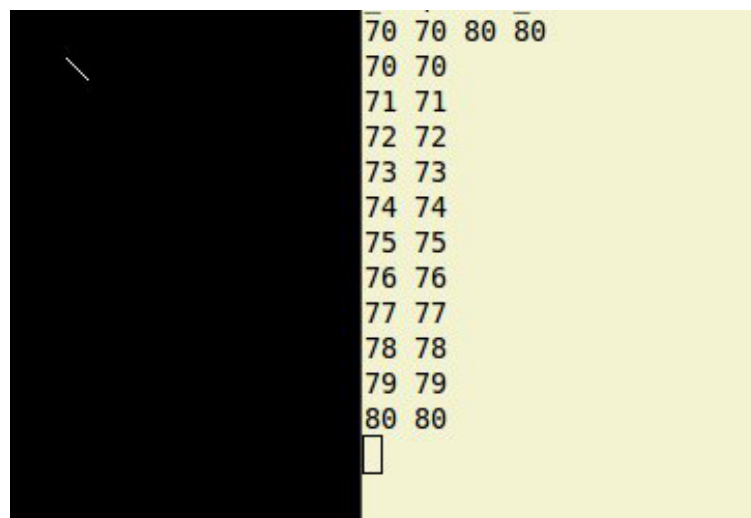
int main(){
    int x1, x2, y1, y2;
    fp = fopen(coordinateFile, "r");
    assert(fp);

    int graphDriver = DETECT, graphMode;
    initgraph(&graphDriver, &graphMode, NULL);

    while (!feof(fp)){
        fscanf(fp, "%d %d %d %d", &x1, &y1, &x2, &y2);
        directDrawLine(x1, y1, x2, y2);
    }

    fclose(fp);
    fprintf(stdout, "MAX : (%d, %d)\n", getmaxx(), getmaxy() );
    delay(10000);
    closegraph();
    return 0;
}

```

**OUTPUT:**

### 3. Write a program to implement Digital Differential Analyzer (DDA) .

Digital Differential Analyzer (DDA) algorithm is the simple line generation algorithm which is explained step by step here.

- 1) Get the input of two end points (X0,Y0)(X0,Y0) and (X1,Y1)(X1,Y1).
- 2) Calculate the difference between two end points.

```
dx = X1 - X0
dy = Y1 - Y0
```

- 3) Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If  $dx > dy$ , then you need more steps in x coordinate; otherwise in y coordinate.

```
if (dx > dy)
    Steps = absolute(dx);
else
    Steps = absolute(dy);
```

- 4) Calculate the increment in x coordinate and y coordinate.

```
Xincrement = dx / (float) steps;
Yincrement = dy / (float) steps;
```

- 5) Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

```
for(int v=0; v < Steps; v++)
{
    x = x + Xincrement;
    y = y + Yincrement;
    putpixel(x,y);
}
```

**Program:**

```

#include <graphics.h>
#include <bits/stdc++.h>
using namespace std;

typedef struct point{
    int x, y;
    point(){}
    point(int x1, int y1){
        x = x1;
        y = y1;
    }
    friend ostream & operator<<(ostream &c, point &a){
        c << "(" << a.x << "," << a.y << ")" ";
        return c;
    }
    bool operator<(point b){
        if (x == b.x && y < b.y)
            return true;
        if (y == b.y && x < b.x)
            return true;
        if (x < b.x && y < b.y)
            return true;
        else return false;
    }
    bool operator==(point b){
        return (x == b.x && y == b.y)?true:false;
    }
}point;

typedef vector<pair<point , point> > lines;

void plotLine(point a, point b){
    // Given a pair of end points a and b, draw a line segment
    int dx = b.x - a.x, dy = b.y - a.y;
    float m;
    if (dx == 0)
        m = INT_MAX * 1.;

```



```

else m = (dy * 1.)/dx;    //slope
float c = a.y - m * a.x;  //y intercept

point start(a.x, a.y);    // Start plotting from first point and repeat for dx times
point end(b.x, b.y);
if (b < a){
    start = point(b.x, b.y);
    end = point(a.x, a.y);
    cout <<"Point b is small. Start with small\n";
}

while(true){
    putpixel(start.x, getmaxy() - start.y, GREEN);
    // cout << " Plotting : " << start << endl;
    if (start == end) break;

    // Move according to values
    if (m == INT_MAX * 1.){
        start.y += 1;
    }
    else if (m <= 1.0){
        start.x += 1;
        start.y = m * start.x + c;
    }
    else{
        start.y += 1;
        start.x = (start.y - c)/m;
    }
}
}

int main(){
    // Initialise graphic drivers and graphModes;
    int graphMode, graphDriver = DETECT;
    int x1, x2, y1, y2;
    initgraph(&graphDriver, &graphMode, NULL);

    // read coordinates from a file
    ifstream inf ("coordinates");

```

```

lines newLines;

while (!inf.eof()){
    inf >> x1 >> y1 >> x2 >> y2 ;
    newLines.push_back(make_pair(point(x1, y1), point(x2, y2) ) );
}
inf.close();

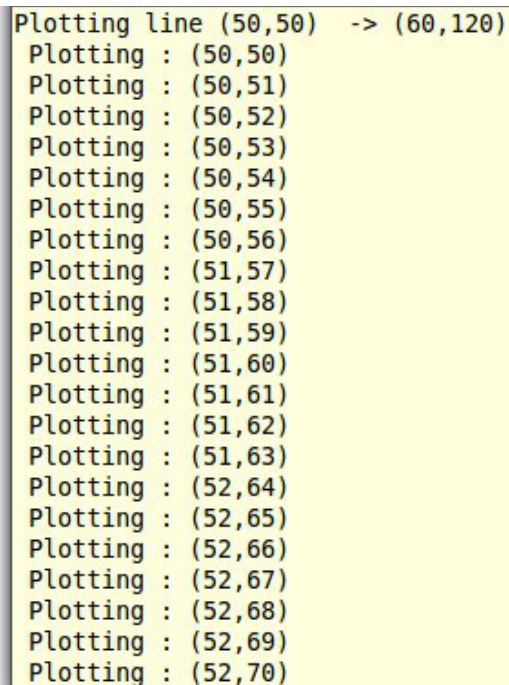
for (auto a : newLines){
    cout <<"\nPlotting line "<<a.first<<" -> "<<a.second<<endl;
    plotLine(a.first, a.second);
}

delay(100000);
closegraph();

return 0;
}

```

### **OUTPUT:**



```

Plotting line (50,50) -> (60,120)
Plotting : (50,50)
Plotting : (50,51)
Plotting : (50,52)
Plotting : (50,53)
Plotting : (50,54)
Plotting : (50,55)
Plotting : (50,56)
Plotting : (51,57)
Plotting : (51,58)
Plotting : (51,59)
Plotting : (51,60)
Plotting : (51,61)
Plotting : (51,62)
Plotting : (51,63)
Plotting : (52,64)
Plotting : (52,65)
Plotting : (52,66)
Plotting : (52,67)
Plotting : (52,68)
Plotting : (52,69)
Plotting : (52,70)

```

4.	<b>Write a program to implement the Bresenham's Line Algorithm</b>
----	--

The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates.

- 1) Input the two end-points of line, storing the left end-point in  $(x_0, y_0)$ .
- 2) Plot the point  $(x_0, y_0)$ .
- 3) Calculate the constants  $dx$ ,  $dy$ ,  $2dy$ , and  $(2dy - 2dx)$  and get the first value for the decision parameter as  $\Rightarrow p_0 = 2dy - dx$
- 4) At each  $x_k$  along the line, starting at  $k = 0$ , perform the following test –
  - If  $p_k < 0$ , the next point to plot is  $(x_{k+1}, y_k)$  and
 
$$p_{k+1} = p_k + 2dy$$
  - Otherwise,
 
$$p_{k+1} = p_k + 2dy - 2dx$$
- 5) Repeat step 4  $(dx - 1)$  times.

**PROGRAM:**

```
#include <graphics.h>
#include <bits/stdc++.h>
using namespace std;

typedef struct point{
    int x, y;
    point(){}
    point(int x1, int y1){
        x = x1;
        y = y1;
    }

    bool operator<(point b){
        if (x < b.x)
            return true;
```

```

        else return false;
    }
}point;

typedef vector<pair<point , point> > lines;

void plotLine(point a, point b){
    // Given a pair of end points a and b, draw a line segment
    int dx = b.x - a.x, dy = b.y - a.y;
    float m;
    if (dx == 0)
        m = INFINITY;
    else m = (dy * 1.)/dx;    //slope

    point start(a.x, a.y);    // Start plotting from first point and repeat for dx times
    point end(b.x, b.y);

    int p = 2 * dy - dx ;
    int k = 0, yk;
    cout <<" \n M: "<<m;

    cout <<"K\tPk\t(x, y):\n";

    // M = infinity
    if (m == INFINITY){
        cout <<"\nm = INF\n";
        while(start.y != end.y){
            cout <<k <<"\t" << p <<"\t" << start << endl;
            start.y += 1;
            putpixel(start.x, getmaxy() - start.y, GREEN);
            continue;
        }
    }
    // For 0 < m < 1, normal algo
    if (m >= 0 && m <= 1){
        cout <<"\n0 <= m <= 1\n";
        if (b.x < a.x){
            start = point(b.x, b.y);
            end = point(a.x, a.y);
        }
    }
}

```

```

    }
    yk = start.y;

    while (start.x <= end.x){
        cout <<k <<"\t" << p <<"\t" << start << endl;
        if (start == end) break;
        start.x += 1;
        start.y = (p > 0) ? start.y + 1: start.y;

        putpixel(start.x, getmaxy() - start.y, BLACK);
        p += 2 *(end.y - start.y) + 2 *(end.x - start.x) *(yk - start.y);
        k ++;
        yk = start.y;
    }
}
// For m > 1, interchange role of x and y
if (m > 1){
    cout <<"\nm > 1\n";
    if (b < a){
        start = point(b.x, b.y);
        end = point(a.x, a.y);
    }

    yk = start.x;
    while (start.y <= end.y){
        cout <<k <<"\t" << p <<"\t" << start << endl;

        if (start == end) break;
        start.y += 1;
        start.x = (p > 0) ? start.x + 1: start.x;

        putpixel(start.x, getmaxy() - start.y, BLACK);
        p += 2 *(end.x - start.x) + 2 *(end.y - start.y) *(yk - start.x);
        k ++;
        yk = start.x;
    }
}
}

```

```

int main(){
    int graphMode, graphDriver = DETECT;
    int x1, x2, y1, y2;
    initgraph(&graphDriver, &graphMode, NULL);

    ifstream inf ("coordinates");

    inf >> x1 >> y1 >> x2 >> y2 ;
    pair<point, point> a = make_pair(point(x1, y1), point(x2, y2));
    cout << "\nPlotting line "<<a.first<<" -> "<<a.second<<endl;
    plotLine(a.first, a.second);
    inf.close();

    closegraph();
    return 0;
}

```

### **OUTPUT:**

```

Plotting line (100,100)  -> (110,114)

M: 1.4K      Pk      (x, y):

m > 1
0      18      (100,100)
1      10      (101,101)
2      2       (102,102)
3      -6      (103,103)
4      8       (103,104)
5      2       (104,105)
6      -4      (105,106)
7      6       (105,107)
8      2       (106,108)
9      -2      (107,109)
10     4       (107,110)
11     2       (108,111)
12     0       (109,112)
13     2       (109,113)
14     2       (110,114)

```

<b>5.</b>	<b>Program to implement the polar co-ordinates method to draw circle</b>
-----------	--

**Description:**

This algorithm is based on the parametric form of the circle equation.

$$x = h + r \cos\theta$$

$$y = k + r \sin\theta$$

where  $r$  is the radius of the circle, and  $h,k$  are the coordinates of the center.

What these equation do is generate the  $x,y$  coordinates of a point on the circle given an angle  $\theta$  (theta). The algorithm starts with theta at zero, and then loops adding an increment to theta each time round the loop. It draws straight line segments between these successive points on the circle. The circle is thus drawn as a series of straight lines. If the increment is small enough, the result looks like a circle to the eye.

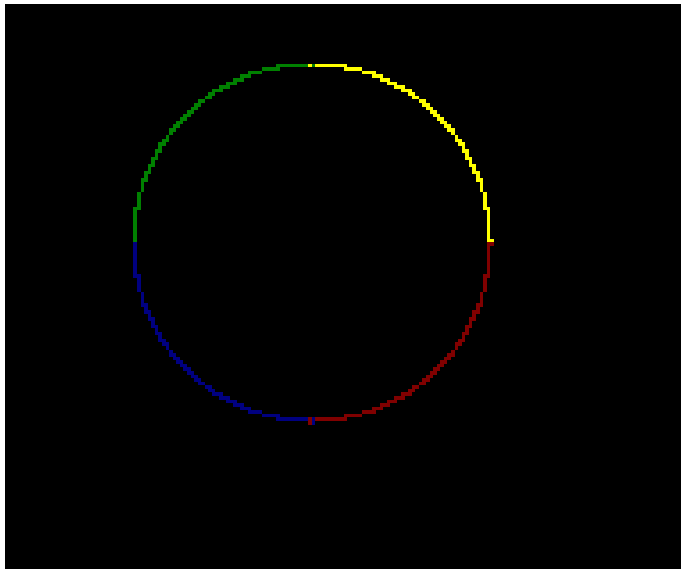
**Program:**

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

int main(){
    int gd=DETECT,gm,i;
    initgraph(&gd,&gm,NULL);
    int xc,yc,r;
    float xi,yi,xf,yf;
    float theta;
    setcolor(4);
    cout<<"Enter (xc,yc) and radius";
    cin>>xc>>yc>>r;
```

```
while(theta<=6.294){  
    putpixel(xf,yf,BLUE);  
    theta=theta+0.01;  
    xf=xc+ (r*cos(theta));  
    yf=yc+ (r*sin(theta));  
}  
delay(20000);  
return 0;  
}
```

**Output:**





<b>6.</b>	<b>Write a program to implement the midpoint circle algorithm.</b>
-----------	--

**Description:**

We cannot display a continuous arc on the raster display. Instead, we have to choose the nearest pixel position to complete the arc. We have put the pixel at (X, Y) location and now need to decide where to put the next pixel – at (X+1, Y) or at (X+1, Y-1). The algorithm is:

1. Input radius **r** and circle center (xc,yc)(xc,yc) and obtain the first point on the circumference of the circle centered on the origin as (x0,y0) = (0,r)

2. Calculate the initial value of decision parameter as :  $P_0 = 5/4 - r$

3. At each XK position starting at K=0, perform the following test :

-> If  $P_K < 0$  then next point on circle (0,0) is  $(X_{K+1}, Y_K)$  and  $P_{K+1} = P_K + 2X_{K+1} + 1$

-> Else  $P_{K+1} = P_K + 2X_{K+1} + 1 - 2Y_{K+1}$

where,  $2X_{K+1} = 2X_{K+2}$  and  $2Y_{K+1} = 2Y_{K-2}$ .

4. Determine the symmetry points in other seven octants.

5. Move each calculate pixel position (X, Y) onto the circular path centered on (XC,YC) and plot the coordinate values.

$$X = X + X_C, \quad Y = Y + Y_C$$

6. Repeat step-3 through 5 until  $X \geq Y$ .

**Program:**

```
#include <graphics.h>
#include <bits/stdc++.h>
using namespace std;
```

```
typedef struct point{
    int x, y, origX, origY;
    point(){}
    point(int x1, int y1){
```

```

        origX = x = x1;
        origY = y = y1;
    }
}point;

typedef vector< pair<point, int> > circles;

void circlePlotPoints(point center, point start, int r){
    int h = center.x, k = center.y;
    int x = start.x, y = start.y;
    putpixel(h + x, getmaxy() - (k + y), BLACK);
    putpixel(h + x, getmaxy() - (k - y), BLACK);
    putpixel(h - x, getmaxy() - (k + y), BLACK);
    putpixel(h - x, getmaxy() - (k - y), BLACK);
    putpixel(h + y, getmaxy() - (k + x), BLACK);
    putpixel(h + y, getmaxy() - (k - x), BLACK);
    putpixel(h - y, getmaxy() - (k + x), BLACK);
    putpixel(h - y, getmaxy() - (k - x), BLACK);
}

void circleMidPoint(point center, int r){
    point start(0, r);
    int p = 1 - r;
    putpixel(center.x, getmaxy() - center.y, LIGHTMAGENTA);

    while (start.x < start.y){
        cout << "Plotting point: " << start << " and P: " << p << endl;
        circlePlotPoints(center, start, r);
        start.x += 1;
        if (p < 0)
            p += 2 * start.x + 1;
        else{
            start.y -= 1;
            p += 2 * (start.x - start.y) + 1;
        }
    }
}

int main(){
    // Initialise graphic drivers and graphModes;
    int graphMode, graphDriver = DETECT;

```

```

int x, y, r;
initgraph(&graphDriver, &graphMode, NULL);
ifstream inf ("coordinates");
circles newCircles;

inf >> x >> y >> r;
newCircles.push_back(make_pair(point(x, y), r ));

inf.close();
circles :: iterator it;

for (it = newCircles.begin(); it != newCircles.end(); it++){
    cout <<"\nPlotting circle "<<it->first<<" with radius -> "<<it->second<<"\n\n";
    circleMidPoint(it->first, it->second);
}

delay(100000);
closegraph();

return 0;
}

```

### **OUTPUT:**



Plotting circle (300,250) with radius -> 20

```

Plotting point: (0,20) and P: -19
Plotting point: (1,20) and P: -16
Plotting point: (2,20) and P: -11
Plotting point: (3,20) and P: -4
Plotting point: (4,20) and P: 5
Plotting point: (5,19) and P: -22
Plotting point: (6,19) and P: -9
Plotting point: (7,19) and P: 6
Plotting point: (8,18) and P: -13
Plotting point: (9,18) and P: 6
Plotting point: (10,17) and P: -7
Plotting point: (11,17) and P: 16
Plotting point: (12,16) and P: 9
Plotting point: (13,15) and P: 6

```

<b>7.</b>	<b>Write a program to implement the ellipse using Bresenham algorithm.</b>
-----------	--

The theory for drawing Ellipse with Bresenham algorithm is same as that of Circle drawing. But the difference is that the Ellipse is divided into two regions because it has two radii. The regions are separated from each other at a point where the slope of the tangent line is  $-1$ . So we need to draw 2 regions in first quadrant and draw in other quadrant symmetrically.

1. Use symmetry of ellipse.

2. Divide the quadrant into two regions, the boundary of two regions is the point at which the curve has a slope of  $-1$ .

3. Process by taking unit steps in the x direction to the point P, then taking unit steps in the y direction.

4. Apply algorithm for both the sections as done in the circle bresenham.

Obtain the formula by putting the following values in the equation of ellipse:

$$p1k = f(xk+1, yk - \frac{1}{2})$$

$$p2k = f(xk + \frac{1}{2}, yk - 1)$$

where  $f(x,y) = (x^2/a^2) + (y^2/b^2) - 1$

**Program:**

```
#include <graphics.h>
#include <bits/stdc++.h>
using namespace std;
```

```
typedef struct point{
    int x, y, origX, origY;
    point(){}
    point(int x1, int y1){
        origX = x = x1;
        origY = y = y1;
    }
}point;
```

```
typedef vector< pair<point, point> > ellipses;
// First pair represents the origin and next gives a, b
```

```

void drawEllipse(point center, point start){
    int h = center.x, k = center.y;
    int x = start.x, y = start.y;
    putpixel(h + x, getmaxy() - (k + y), BLUE);
    putpixel(h + x, getmaxy() - (k - y), BLUE);
    putpixel(h - x, getmaxy() - (k + y), BLUE);
    putpixel(h - x, getmaxy() - (k - y), BLUE);
}

```

```

void ellipseBresen(point origin, point para){
    int a = para.x, b = para.y;
    // int xA = origin.x, yA = origin.y;
    // ellipse(xA, yA, 0, 360, b, a);
    int a2 = a * a, b2 = b * b;
    int x = 0, y = b;
    float d = b2 - (a2 * b) + a2/4.;

```

```

// REGION I

```

```

while (b2*x <= a2*y){
    point zy = point(x, y);
    cout << "\nREG I: " << zy << "\td: " << d;
    drawEllipse(origin, zy);
    x ++;
    if (d > 0){
        y--;
        d += 2.0 * b2 * x + b2 - 2.0 * a2 * y;
    }
    else d += b2 + 2.0 * b2 * x;
}

```

```

// REGION II

```

```

x = a, y = 0;
d = a2 + 2.0 * b2 * a + b2/4.;

while (a2*y <= b2*x){
    point zy = point(x, y);
    cout << "\nREG II: " << zy << "\td: " << d;
    drawEllipse(origin, zy);
    y++;
}

```

```

    if (d > 0)
        d += a2 - 2.0 * a2 * y;

    else{
        x--;
        d += 2.0 * b2 * x - 2.0 * a2 * y + a2;
    }
}
}

int main(){

    int graphMode, graphDriver = DETECT;
    int x, y , a, b;
    initgraph(&graphDriver, &graphMode, NULL);

    // read coordinates from a file
    ifstream inf ("coordinates");
    ellipses newEllipse;

    while (!inf.eof()){
        inf >> x >> y >> a >> b;
        newEllipse.push_back(make_pair(point(x, y), point(a, b) ));
    }
    inf.close();
    ellipses :: iterator it;

    for (it = newEllipse.begin(); it != newEllipse.end(); it++){
        cout << "\nPlotting ellipse "<< it -> first << " with a, b-> "<< it -> second << "\n\n";
        ellipseBresen(it -> first, it -> second);
    }

    delay(100000);
    closegraph();
    return 0;
}

```

**OUTPUT:**

Plotting ellipse (400,100) with a, b-> (60,70)

REG I: (0,70)	d: -246200
REG I: (1,70)	d: -231500
REG I: (2,70)	d: -207000
REG I: (3,70)	d: -172700
REG I: (4,70)	d: -128600
REG I: (5,70)	d: -74700
REG I: (6,70)	d: -11000
REG I: (7,70)	d: 62500
REG I: (8,69)	d: -351000
REG I: (9,69)	d: -257900
REG I: (10,69)	d: -155000
REG I: (11,69)	d: -42300
REG I: (12,69)	d: 80200
REG I: (13,68)	d: -277100
REG I: (14,68)	d: -135000
REG I: (15,68)	d: 16900

<b>8.</b>	<b>Write a program to perform 2D transformations such as translation, rotation, scaling, reflection and shearing.</b>
-----------	---

**Description:**

Fundamental to all computer graphics system is the ability to simulate the manipulation of objects in space. This simulated spatial manipulation is referred to as transformation. The need for transformation arises when several objects, each of which is independently defined in its own co-ordinate system, need to properly positioned into a common scene in a master co-ordinate system. Transformations are also relevant in other areas of image synthesis process. The algorithm for its implementation is as follows:

1. Start the program.

2. Input the object coordinates

3. For Translation:

- a) Enter the translation factors  $t_x$  and  $t_y$ .
- b) Move the original coordinate position  $(x, y)$  to a new position  $(x_1, y_1)$ . i.e.  $x = x + t_x$ ,  $y = y + t_y$ .
- c) Display the object after translation.

4. For Rotation:

- a) Enter the radian for rotation angle  $\theta$ .
- b) Rotate a point at position  $(x, y)$  through an angle  $\theta$  about the origin:
 

$$x_1 = x \cos \theta - y \sin \theta$$

$$y_1 = y \cos \theta + x \sin \theta$$
- c) Display the object after rotation.

5. For Scaling:

- a) Input the scaled factors  $s_x$  and  $s_y$ .
- b) The transformed coordinates  $(x_1, y_1)$ ,  $x_1 = x \cdot s_x$  and  $y_1 = y \cdot s_y$ .
- c) Display the object after scaling

6. For Shearing:

- a) Input the shearing factors  $sh_x$  and  $sh_y$ .



- b) Shearing related to x axis : Transform coordinates  $x1=x+shx*y$  and  $y1=y$ .
- c) Shearing related to y axis : Transform coordinates  $x1=x$  and  $y1=y+shy*x$ .
- d) Input the xref and yref values.
- e) X axis shear related to the reference line  $y-yref$  is  $x1=x+shx(y-yref)$  and  $y1=y$ .
- f) Y axis shear related to the reference line  $x=xref$  is  $x1=x$
- g) Display the object after shearing

8. Stop the Program.

**Program:**

```
#include <bits/stdc++.h>
#include <graphics.h>
using namespace std;

/* A program to perform 2d transformations on a figure from a file
Line 1: Coordinations of the given figure
Line 2: Translation of figure in three direction x, y, z
Line 3: Rotation of triangle in three directions x, y, x
Line 4: Scaling of triangle in x and y directions;
*/
typedef struct point{
    int x, y;
    point(){};
    point(int a, int b){
        x = a;
        y = b;
    }
}point;

typedef struct translation{
    int x, y;
    translation(){}
}translation;

typedef struct rotation{
    int angle, fixedX, fixedY;
```

```

}rotation;

typedef struct scale{
    float x, y;
    int fixedX, fixedY;
}scale;

typedef struct shear{
    float val;
    int shX, shY, axis;
}shear;

void plotTriangle(point a, point b, point c){
    //Given three coordinates plot the triangle resulting from them
    line(a.x, a.y, b.x, b.y);
    line(a.x, a.y, c.x, c.y);
    line(b.x, b.y, c.x, c.y);
}

point translatePoint(point a, translation t){
    return point(a.x + t.x, a.y + t.y);
}

point scalePoint(point a, scale s){
    return point(a.x * s.x + s.fixedX * (1 - s.x), a.y * s.y + s.fixedY * (1 - s.y));
}

point rotatePoint(point a, rotation r){
    r.angle = (M_PI * r.angle)/180;
    double x = r.fixedX + (a.x - r.fixedX)* cos(r.angle) - (a.y - r.fixedY) * sin(r.angle);
    double y = r.fixedY + (a.y - r.fixedY)* sin(r.angle) - (a.x - r.fixedX) * cos(r.angle);
    return point(x + a.x, y + a.y);
}

point applyShear(point a, shear h){
    if (h.axis == 1)
        return point(a.x + h.val * (a.y - h.shY), a.y);
    else
        return point(a.x, a.y + h.val * (a.x - h.shX));
}

```

```

int main(){
    int graphMode, graphDriver = DETECT;

    point a, b, c;
    translation t;
    rotation r;
    scale s;
    shear h;

    cout << "\n Enter coordinates of triangle: ";
    cin >> a.x >> a.y >> b.x >> b.y >> c.x >> c.y;

    cout << "\n Enter translation in x and y axis: ";
    cin >> t.x >> t.y;

    cout << "\n Enter angle of rotation as well as point: ";
    cin >> r.angle >> r.fixedX >> r.fixedY;

    cout << "\n Enter scaling factor in x and y and a point: ";
    cin >> s.x >> s.y >> s.fixedX >> s.fixedY ;

    cout << "\n Enter shear value, point and axis (x = 1 | y = 0): ";
    cin >> h.val >> h.shX >> h.shY >> h.axis;

    initgraph(&graphDriver, &graphMode, NULL);

    // Plot initial triangle
    plotTriangle(a, b, c);

    // Translate the triangle
    point transA = translatePoint(a, t);
    point transB = translatePoint(b, t);
    point transC = translatePoint(c, t);

    //Rotate the triangle
    point rotA = rotatePoint(a, r);
    point rotB = rotatePoint(b, r);
    point rotC = rotatePoint(c, r);

```

```

printf("\n\n\tROTATED TRIANGLE\n");
plotTriangle(a, b, c);
plotTriangle(rotA, rotB, rotC);

//Scale the triangle
point scaleA = scalePoint(a, s);
point scaleB = scalePoint(b, s);
point scaleC = scalePoint(c, s);

printf("\n\n\tSCALED TRIANGLE\n");
plotTriangle(a, b, c);
plotTriangle(scaleA, scaleB, scaleC);

//Shear the triangle
point shearA = applyShear(a, h);
point shearB = applyShear(b, h);
point shearC = applyShear(c, h);

printf("\n\n\tSHEARED TRIANGLE\n");
plotTriangle(a, b, c);
plotTriangle(shearA, shearB, shearC);

delay(10000);
closegraph();
return 0;
}

```

### **OUTPUT:**

```

aman@aman ~/Desktop/prog/Graphics/3-172D_TRANS $ ./2dTransform

Enter coordinates of triangle: 30 150 10 200 60 200

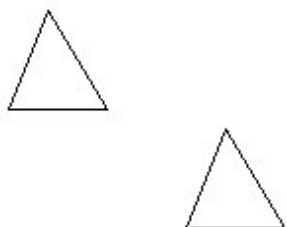
Enter translation in x and y axis: 90 60

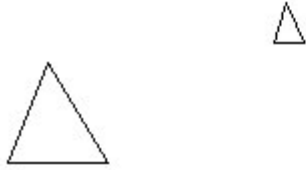
Enter angle of rotation as well as point: 90 100 200

Enter scaling factor in x and y and a point: 0.3 0.4 100 200

Enter shear value, point and axis (x = 1 | y = 0): 3 50 100 1

```

**TRANSLATION:*****TRANSLATED TRIANGLE*****ROTATION:*****ROTATED TRIANGLE***

**SCALING:****SCALED TRIANGLE****SHEARING:****SHEARED TRIANGLE**