নোয়াখালী বিজ্ঞান ও প্রযুক্তি বিশ্ববিদ্যালয়
Noakhali Science and Technology University
Noakhali-3814

ইনফরমেশন এন্ড কমিউনিকেশন ইঞ্জিনিয়ারিং বিভাগ
Department of Information and Communication Engineering

Course Title:

Computer Graphics Lab

Course code:

ICE-3210

Lab Report on:

Computer Graphics

Submitted to:

Md. Bipul Hossain
Lecturer
Dept. of ICE, NSTU.

Submitted by:

Abdul Awal Nadim
Roll: ASH1811056M
Session: 2017-18

Submission date: 14-02-2022

1. Name Of Program

Write a program to draw a Hut or other geometrical figures

Theoretical Background

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 We 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.

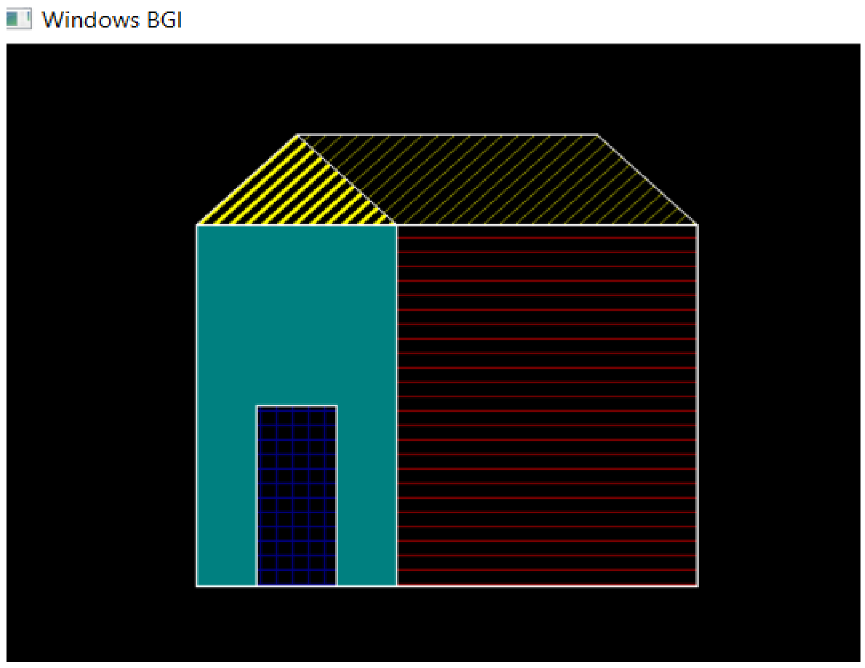
3.The setfillstyle() function which sets any fill pattern in any shape created in C program using graphics.

4.The floodfill() function is used to fill an enclosed area with any color.

Program

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

int main(){
 int gd = DETECT,gm;
    initgraph(&gd, &gm, "");
    /* Draw Hut */
    delay(500);
    rectangle(100,100,200,300);
    rectangle(200,100,350,300);
    rectangle(130,200,170,300);
    line(100,100,150,50);
    line(150,50,200,100);
    line(150,50,300,50);
    line(300,50,350,100);


    setfillstyle(SOLID_FILL, 3);
    floodfill(152, 182, WHITE);

    setfillstyle(LINE_FILL, 4);
    floodfill(252, 182, WHITE);

    setfillstyle(HATCH_FILL, 1);
    floodfill(131, 201, WHITE);

    setfillstyle(SLASH_FILL, 14);
    floodfill(150, 51, WHITE);

    setfillstyle(LTSLASH_FILL, 6);
    floodfill(300, 51, WHITE);

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

<u>Input & Output</u>



2. Name Of Program

Write a program to draw a line using Bresenhem's Algo.

Theoretical Background

1. Start.

2. Declare variables x,y,x1,y1,x2,y2,p,dx,dy and also declare gdriver=DETECT, gmode.

3. Initialize the graphic mode with the path location in TC folder.

4. Input the two line end-points and store the left end-points in (x1, y1).

5. Load (x1, y1) into the frame buffer; that is, plot the first point put x=x1, y=y1.

6. Calculate dx=x2-x1 and dy=y2-y1, and obtain the initial value of decision parameter p as p = (2dy-dx).

7. Starting from first point (x,y) perform the following test:

8. Repeat step 9 while(x<=x2).

9. If p<0, next point is (x+1,y) and p=(p+2dy).

10. Otherwise, the next point to plot is (x+1,y+1) and p=(p+2dy-2dx).

11. Place pixels using putpixel at points (x,y) in specified colour.

12. Close Graph.

13. Stop

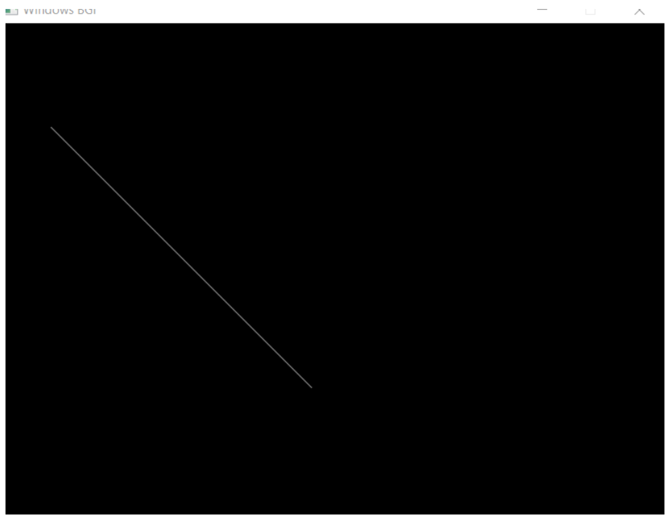
Program

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

```
void drawline(int x0, int y0, int x1, int y1)
{
    int dx, dy, p, x, y;
    dx=x1-x0;
    dy=y1-y0;
    x=x0;
    y=y0;
    p=2*dy-dx;
    while(x<=x1)
    {
        delay(50);
        if(p>=0)
        {
            putpixel(x,y,7);
            y=y+1;
            p=p+2*dy-2*dx;
        }
        else
        {
            putpixel(x,y,7);
            p=p+2*dy;
        }
            x=x+1;

    }

        cout<<x<<" "<<y<<endl;


}

int main()

{

    int x0, y0, x1, y1;



    int gd = DETECT, gm;
          initgraph(&gd, &gm, "");
    printf("Enter co-ordinates of first point: ");
    scanf("%d%d", &x0, &y0);
    printf("Enter co-ordinates of second point: ");
    scanf("%d%d", &x1, &y1);

    drawline(x0, y0, x1, y1);

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

<u>Input & Output</u>

```
Enter co-ordinates of first point: 50 100
Enter co-ordinates of second point: 300 400
```



3. <u>Name Of Program</u>

Write a program to draw a line using DDA algorithm.

<u>Theoretical Background</u>

1. Start.

2. Declare variables x,y,x1,y1,x2,y2,k,dx,dy,s,xi,yi and also declare gdriver=DETECT,gmode.

3. Initialise the graphic mode with the path location in TC folder.

4. Input the two line end-points and store the left end-points in (x1,y1).

5. Load (x1,y1) into the frame buffer;that is,plot the first point.put x=x1,y=y1.

6. Calculate dx=x2-x1 and dy=y2-y1.

7. If abs(dx) > abs(dy), do s=abs(dx).

8. Otherwise s= abs(dy).

9. Then xi=dx/s and yi=dy/s.

10. Start from k=0 and continuing till k<s,the points will be
    i. x = x+xi.
    ii. Y = y+yi.

11. Place pixels using putpixel at points (x,y) in specified colour.

12. Close Graph.

13. Stop

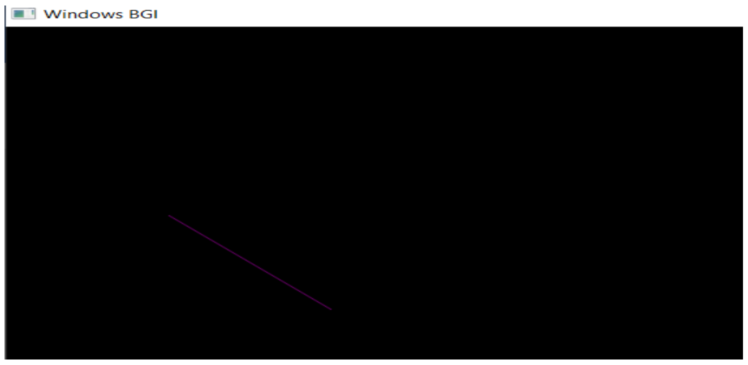
Program

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

int main( )
{
   float x, y, x1, y1, x2, y2, dx, dy, step;
   int gd = DETECT;
   int gm;

   initgraph(&gd, &gm, "DDA Algorithm");
   cout << "Enter the value of x1 and y1: ";
   cin >> x1 >> y1;
```

```
        cout << "Enter the value of x2 and y2: ";
        cin >> x2 >> y2;

        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;

        for(int i = 1; i <= step; i++){
            delay(100);
            putpixel(x, y, 5);
            x = x + dx;
            y = y + dy;

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

Input & Output

```
Enter the value of x1 and y1: 100 200

Enter the value of x2 and y2: 200 100
```



4. Name Of Program

Write a program to draw a line using Mid-Point algorithm.

Theoretical Background

In **figure 1**, we have a previous point K (x_k,y_k), and there are two next points; the lower point B (x_k + 1,y_k) and the above point AB **(**x_k + 1,y_k + 1). If the midpoint **m** is below the line, then we select point **AB.**

Let us assume, we have two points of the line = **(x_1, x_2)** and **(y_1, y_2).**

Here, it is ($x_1 < x_2$)
The Linear equation of a line is:
(x, y) = ax + by +c = 0 …………… (1)
$d_x = x_2 - x_1$
$d_y = y_2 - y_1$
As we know, the equation of simple line is:
y = mx +B

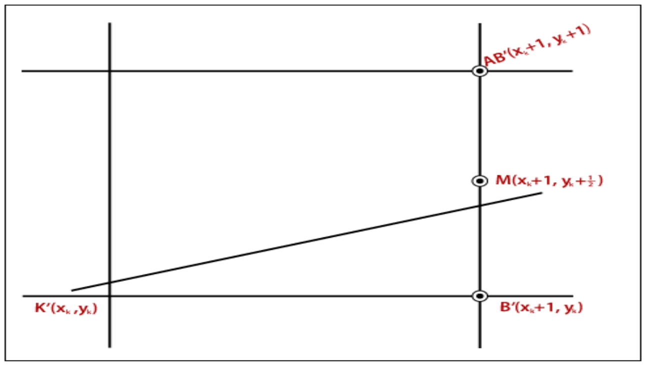


figure : 1

Here, m = dy/dx, we can write it as y = (dy/dx) x +B
Put the value of y in above equation, we get
(x, y) = dy (x) – (dx) y +B. dx = 0…………… (2)
By comparing equation (1) and (2)
a = dy
b = -dx
c = B. dx

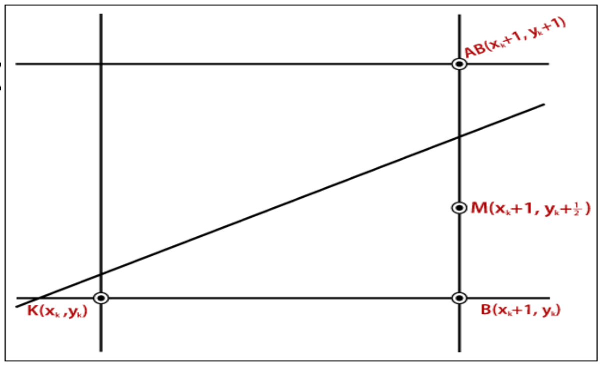


figure : 2

There are some conditions given-
Condition 1: If all the points are on the line, then (x, y) = 0
Condition 2: If all the points are on the line, then (x, y) = Negative Number
Condition 3: If all the points are on the line, then (x, y) = positive Number
Now, m = (x_k +1, y_k + 1/2)

$d = m = (x_k + 1, y_k + 1/2)$

Now, we calculate the distance value **(d).** Here, we discuss two cases:

Case 1: If

We select the lower point B

Then

The value of d is negative ($d < 0$)

We calculate the new and old value of **d-**

$d_n = (x_k + 2, y_k + 1/2)$ {The value of x coordinate is incremented by 1}

We put the value of **(x, y)** in equation **(1)** andwe got the value of $\mathbf{d_o}$

$d_n = a (x_k + 2) + b (y_k + 3/2) + c$ {New Value of d}

$d_o = a (x_k + 1) + b (y_k + 1/2) + c$ {Old Value of d}

Thus,

$?d = d_n – d_o$

$= a (x_k+2) + b (y_k +3/2) + c – a (x_k+1) – b (y_k +1/2) – c$

$= a + b$

$?d = d_n – d_o$

$?d = d_o + a + b$

$?d = d_o + dy – dx$ {$a + (-b) = dy- dx$}

Now, we calculate the initial decision parameter $(\mathbf{d_i})$

$d_i = (x_1 + 1, y_1 + 1/2)$

Put the value of $\mathbf{d_i}$ in

equation **(1)**

$d_i = a (x_1 + 1) + b (y_1 + 1/2) + c$

$= ax_1 + a + by_1 + b/2 + c$

$= (x_1, y_2) + a + b/2$

$= 0 + a + b/2$

$d_i = dy – dx/2$

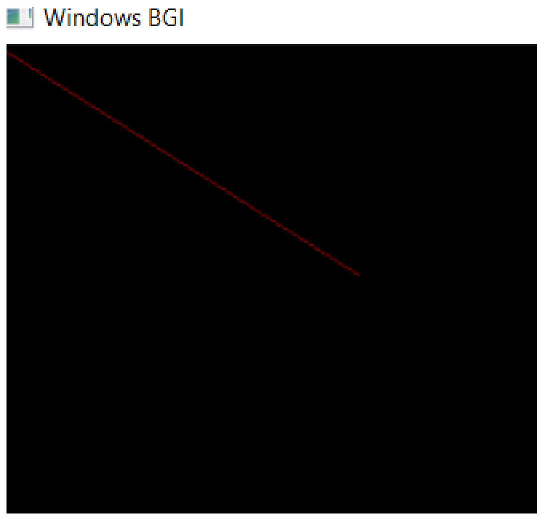
Program

```
#include<bits/stdc++.h>

#include<conio.h>

#include<graphics.h>

using namespace std;

void midPoint(int X1, int Y1, int X2, int Y2)

{
```

```
int dx = X2 - X1;
int dy = Y2 - Y1;

if(dy<=dx)
{
    int d = dy - (dx/2);
    int x = X1, y = Y1;
    putpixel(x,y,RED);
    while (x < X2)
    {
        x++;
        if (d < 0)
            d = d + dy;
        else
        {
            d += (dy - dx);
            y++;
        }
        putpixel(x,y,RED);
    }
}

else if(dx<dy)
{
    int d = dx - (dy/2);
    int x = X1, y = Y1;
    putpixel(x,y,RED);
    while (y < Y2)
    {
        y++;
        if (d < 0)
            d = d + dx;
        else
        {
            d += (dx - dy);
            x++;
        }
        putpixel(x,y,RED);
    }
    }
}
int main()
{
    int gd = DETECT, gm;
    initgraph (&gd, &gm, "");


    int X1, Y1, X2, Y2;
    cout << "Enter the value of x1 and y1: ";
    cin >> X1 >> Y1;
    cout << "Enter the value of x2 and y2: ";
    cin >> X2 >> Y2;
    midPoint(X1, Y1, X2, Y2);

    getch();
    closegraph();

    return 0;
}
```

<u>Input & Output</u>

```
Enter the value of x1 and y1: 2 2
Enter the value of x2 and y2: 180 115
```



5. <u>Name Of Program</u>

Write a program to draw a circle using mid-point algorithm.

Theoretical Background

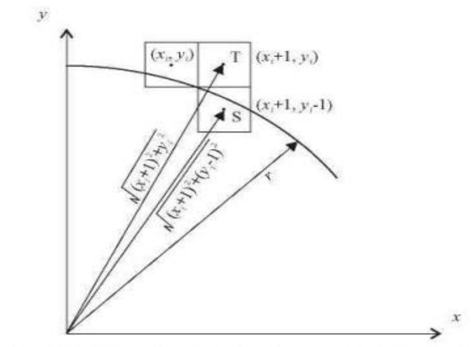


Fig. 3-8 Choosing pixels in Bresenham's circle algorithm.

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:

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

If P_i is -ve ⟹midpoint is inside the circle and we choose pixel T

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

The decision parameter for the next step is:

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

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

$P_{i+1} - P_i = ((xi + 1) + 1)^2 - (xi + 1)2+(y_{i+1-\frac{1}{2}})^2 -(y_{i-\frac{1}{2}})$

$=x_i^2+4+4x_i-x_i^2+1-2x_i+y_{i+1}^2+\frac{1}{4} - y_{i+1} - y_i^2-\frac{1}{4} - y_i$

$=2(xi+1) +1+(y_{i+1}^2 - y_i^2) -(y_{i+1} - y_i)$

$P_{i+1} = P_i + 2(xi+1) + 1 + ({y_{i+1}}^2 - {y_i}^2) - (y_{i+1} - y_i)$………………………equation(4)

If pixel T is choosen $\Longrightarrow P_i<0$

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

If pixel S is choosen $\Longrightarrow P_i \geq 0$

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

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

The following is a description of this midpoint circle algorithm that generates the pixel coordinates in the 90° to 45° octant:

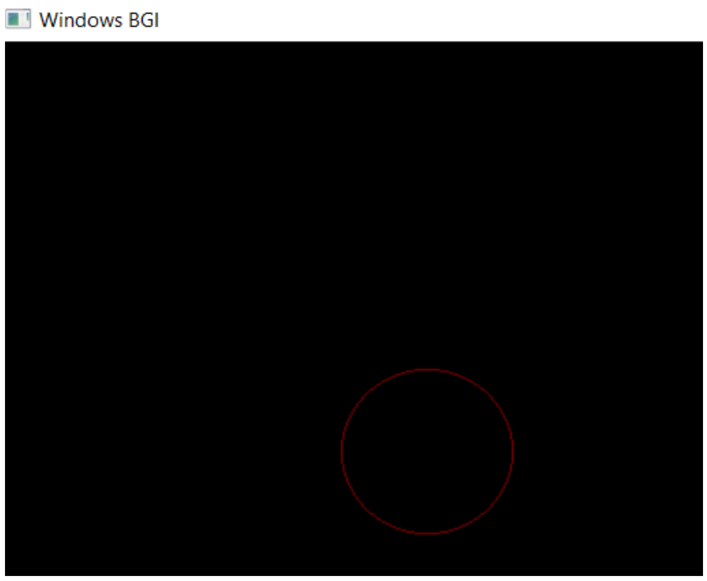
```
Int x=0, y=r, p=1-r;
While(x<=y) {
setPixel(x,y);
if(p<0)
   p=p+2x+3;
else {
p=p+2(x-y)+5;
y--;
}
X++;
}
```

Program

```
#include <stdio.h>
#include <dos.h>
#include <graphics.h>
#include<bits/stdc++.h>
using namespace std;
void drawCircle(int xc, int yc, int x, int y)
{
        putpixel(xc+x, yc+y, RED);
        putpixel(xc-x, yc+y, RED);
putpixel(xc+x, yc-y, RED);
        putpixel(xc-x, yc-y, RED);
        putpixel(xc+y, yc+x, RED);
        putpixel(xc-y, yc+x, RED);
        putpixel(xc+y, yc-x, RED);
        putpixel(xc-y, yc-x, RED);
}
void circlemid(int xc, int yc, int r)
{
        int x = 0, y = r,p=1-r;
        while(x<=y)
{
    delay(50);
    drawCircle(xc,yc,x,y);
    if(p<0)
        p= p+2*x+3;
    else
    {
        p= p + 2*(x-y)+ 5;
        y--;
    }
    x++;
}
}
int main()
{
        int xc = 250, yc = 250, r ;

        int gd = DETECT, gm;
        initgraph(&gd, &gm, ""); // initialize graph
 cout<<"Enter the radious :";
 cin>>r;
        circlemid(xc, yc, r); // function call
        //delay(500);
        getch();
closegraph();
        return 0;
}
```

Input & Output

Enter the radius : 50



6. Name Of Program

Write a program to draw an Ellipse using Mid-Point algorithm.

Theoretical Background

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)=$b^2x^2 + a^2\ y^2$ - a^2b^2

Now divide the elliptical curve from (0, b) to (a, 0) into two parts at point Q where the slope of the curve is -1.

Slope of the curve is defined by the f(x, y) = 0 $\frac{\Delta y}{\Delta x} = \frac{fx}{fy}$ is where fx & fy are partial derivatives of f(x, y) with respect to x & y.

We have fx = 2b^2 x, fy=2a^2 y & $\frac{dy}{dx} = -\frac{2b^2x}{2a^2y}$ Hence we can monitor the slope value during the scan conversion process to detect Q. Our starting point is (0, b)

Suppose that the coordinates of the last scan converted pixel upon entering step i are (x_i,y_i). We are to select either T (x_{i+1}),y_i) or S (x_{i+1},y_{i-1}) to be the next pixel. The midpoint of T & S is used to define the following decision parameter.

$Pi=f(x_{i+1}),y_i-\frac{1}{2})$

$pi=b^2\ (x_{i+1})^2+a^2\ (y_i-\frac{1}{2})^2-a^2\ b^2$

If pi<0, the midpoint is inside the curve and we choose pixel T.

If pi>0, the midpoint is outside or on the curve and we choose pixel S.

Decision parameter for the next step is:

$p_{i+1}=f(x_{i+1}+1,y_{i+1}-\frac{1}{2})$

$\quad = b^2\ (x_{i+1}+1)^2+a^2\ (y_{i+1}-\frac{1}{2})^2-a^2\ b^2$

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

p_{i+1}-pi=$b^2[((x_{i+1}+1)^2+a^2 (y_{i+1}-\frac{1}{2})^2-(y_i - \frac{1}{2})^2]$

p_{i+1}= pi+2$b^2 x_{i+1}+b^2+a^2 [(y_{i+1}-\frac{1}{2})^2-(y_i - \frac{1}{2})^2]$

If T is chosen pixel (pi<0), we have y_{i+1}=yi.

If S is chosen pixel (pi>0) we have y_{i+1}=yi-1. Thus we can express, p_{i+1} in terms of pi and (x_{i+1},y_{i+1}):

p_{i+1}= pi+2$b^2 x_{i+1}+b^2$

if pi<0

p_{i+1}= pi+2$b^2 x_{i+1}+b^2-2a^2 y_{i+1}$, if pi>0

The initial value for the recursive expression can be obtained by the evaluating the original definition of pi with (0, b):

p1 = $(b^2+a^2 (b-\frac{1}{2})^2-a^2 b^2$

  = $b^2-a^2 b+a^2/4$

Suppose the pixel $(x_j\ y_j)$ has just been scan converted upon entering step j. The next pixel is either U (x_j , y_j-1) or V (x_j+1,y_j-1). The midpoint of the horizontal line connecting U & V is used to define the decision parameter:

$q_j=f(x_j+\frac{1}{2},y_j-1)$

$q_j=b^2 (x_j+\frac{1}{2})^2+a^2 (y_j -1)^2-a^2 b^2$

If $q_j<0$, the midpoint is inside the curve and we choose pixel V.

If $q_j\geq0$, the midpoint is outside the curve and we choose pixel U.Decision parameter for the next step is:

$q_{i+1}=f(x_{j+1}+\frac{1}{2},y_{j+1}-1)$

$= b^2 (x_{j+1}+\frac{1}{2})^2+ a^2 (y_{j+1}-1)^2- a^2 b^2$

Since $y_{i+1}=y_i$-1,we have

$q_{i+1}-q_i=b^2 [(x_{i+1}+\frac{1}{2})^2-(x_j +\frac{1}{2})^2]+a^2 (y_{i+1}-1)^2-(y_{j+1})^2]$

$q_{i+1}=q_j+b^2 [(x_{j+1}+\frac{1}{2})^2-(x_j +\frac{1}{2})^2]-2a^2 y_{j+1}+a^2$

If V is chosen pixel (qj<0), we have x_{j+1}=xj.

If U is chosen pixel (pi>0) we have x_{j+1}=xj. Thus we can express

q_{j+1} in terms of q_j and (x_{j+1}, y_{j+1}):

$q_{i+1} = q_i+2b^2 x_{i+1}-2a^2 y_{i+1}+a^2$ if qj < 0
$=q_j-2a^2 y_{j+1}+a^2$ if qj>0

The initial value for the recursive expression is computed using the original definition of qj. And the coordinates of $(x_k\ y_k)$ of the last pixel choosen for the part 1 of the curve:

$q1 = f(x_k+\frac{1}{2},y_k-1)=b^2 (x_k+\frac{1}{2})^2-a^2 (y_k-1)^2- a^2 b^2$

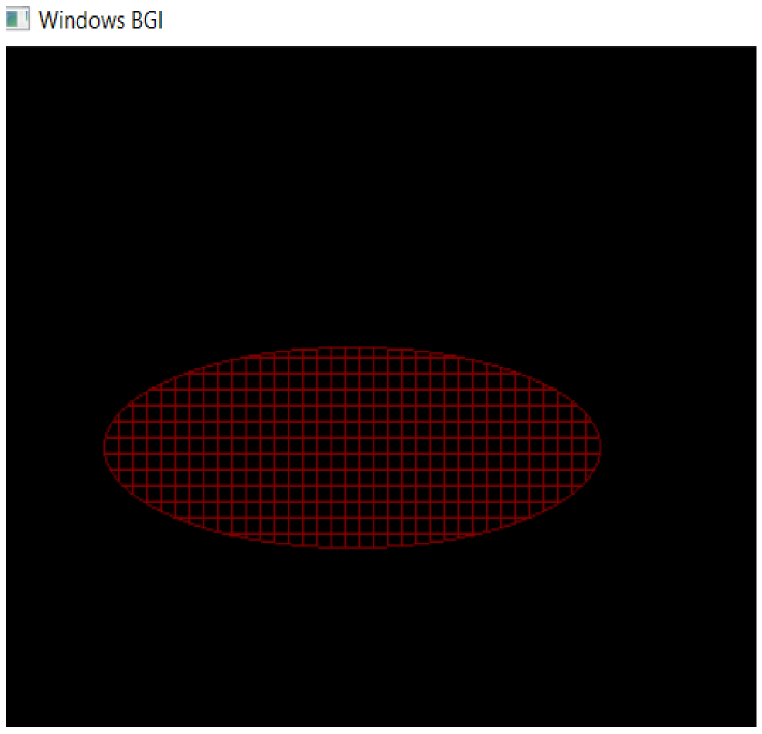
```
int x=0,y=b;
int aa=a*a,bb=b*b,aa2=aa*2,bb2=bb*2;
int fx=0,fy=aa2*b;
int p=bb-aa*b+.25*aa;
while(fx<fy){
   setPixel(x,y);
x++;
fx=fx+bb2;
if(p<0)
  p=p+fx+bb;
else {
  y--;
fy=fy-aa2;
p=p+fx+bb-fy;
   }
}
setPixel(x,y);
p=bb(x+0.5)(x+0.5)+aa(y-1)(y-1)-aa*bb;
while(y>0){
  y--;
fy=fy-aa2;
if(p>=0)
      p=p-fy+aa;
else {
      x++;
     fx=fx+bb2;
     p=p+fx-fy+aa;
    }
   setPixel(x,y);
}
```

Program

```
#include <stdio.h>
#include <dos.h>
#include <graphics.h>
#include<bits/stdc++.h>
using namespace std;
void drawellips(int xc, int yc, int x, int y)
{
    putpixel(xc+x, yc+y, RED);
        putpixel(xc-x, yc+y, RED);
        putpixel(xc+x, yc-y, RED);
        putpixel(xc-x, yc-y, RED);
}

void ellipsedraw(int xc, int yc, int a, int b)
{
        int x=0,y=b;
        int fx=0,fy=2*a*a*b;
        int p=b*b - a*a*b + 0.25 * a* a;
        cout<<"fx "<<fx<<" "<<fy<<endl;
        while(fx<fy) // |slope| <1
    {
        delay(50);
        drawellips(xc,yc,x,y);
        fx+=2*b*b;
        if(p<0)
        {
            p+=2*b*b*x+3*b*b;
        }
        else
        {
            fy-=2*a*a;
            p+=2*b*b*x+3*b*b-(2*a*a*(y-1));
            y--;
        }
        x++;
    }
    int q=(b*b*(x+0.5)*(x+0.5)) + (a*a*(y-1)*(y-1))- a*a*b*b;
    while(y>=0)
    {
        delay(50);
        drawellips(xc,yc,x,y);
        if(q<0)
        {
            q+= 2*b*b*x - 2*a*a*y + 2*b*b + 3*a*a;
            x++;
        }
        else
        {
            q+=3*a*a - 2*a*a*y;
        }
        y--;
    }

}

int main()
{
          int xc = 200, yc = 200, a,b;
          int gd = DETECT, gm;
          initgraph(&gd, &gm, "");
    cout<<"Enter the length of the major axis: ";
    cin>>a;
    cout<<"Enter the length of the minor axis: ";
    cin>>b;

        ellipsedraw(xc, yc, a, b); // function call


    setfillstyle(HATCH_FILL, RED);
    floodfill(xc, yc, RED);

        //delay(500);
        getch();
    closegraph();
        return 0;
}
```

<u>Input & Output</u>

```
Enter the length of the major axis: 140
Enter the length of the minor axis: 50
fx 0 1960000
```



7. <u>Name Of Program</u>

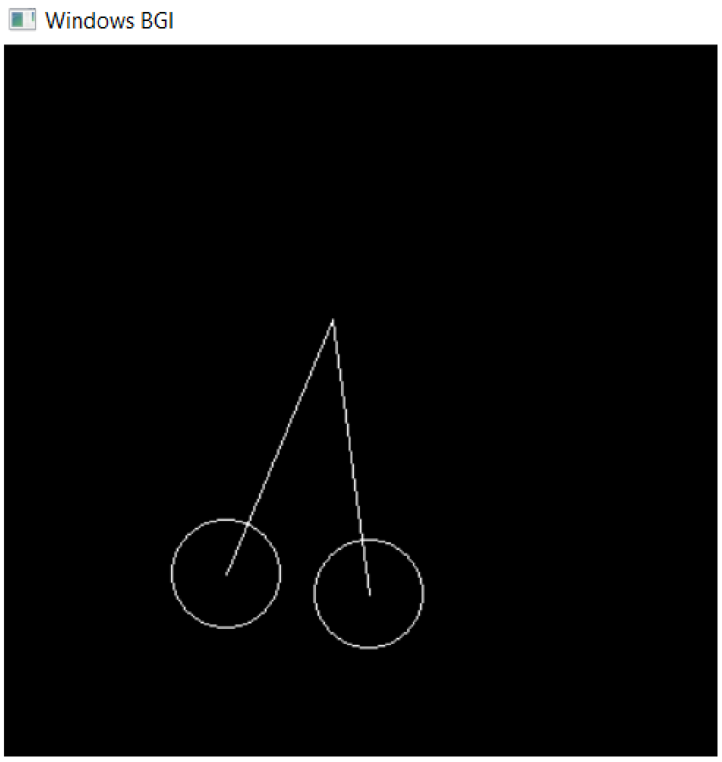
Write a program to rotate a Circle around any arbitrary point or around the boundary of another circle.

Program

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


int npx,npy;
void rotate_point(float cx,float cy,float angle,int px,int py)
{
   float s = sin(angle*3.14/180);
   float c = cos(angle*3.14/180);

   // translate point back to origin:
   px -= cx;
   py -= cy;

   // rotate point
   float xnew = px * c - py * s;
   float ynew = px * s + py * c;

   // translate point back:
   px = xnew + cx;
   py = ynew + cy;

   npx=px;
   npy=py;
}

int main()
{
   int gd = DETECT,gm;
   initgraph(&gd, &gm, "");
   int cx=180,cy=150,angle=30;
   int px=200,py=300,r=30;
   line(cx,cy,px,py);
   circle(px,py,r);

   rotate_point((float)cx,(float)cy,(float)angle,px,py);
   //initgraph(&gd, &gm, "");
   line(cx,cy,npx,npy);
   circle(npx,npy,r);

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

Input & Output

```
180 150 30
200 300 30
```



8. Name Of Program

Write a menu driven program to rotate, scale and translate a line point, square, triangle about the origin

Theoretical Background

. Translation

In translation, an object is displaced a given distance and direction from its original position. If the displacement is given by the vector v = t_xI +t_yJ, the new object point P'(x',y') can be found by applying the transformation TY to P(x, y) (see Fig. 4-2).

$$P' = \boldsymbol{T_v}(P)$$

where x' = x + tx and $y' = y + t_y$

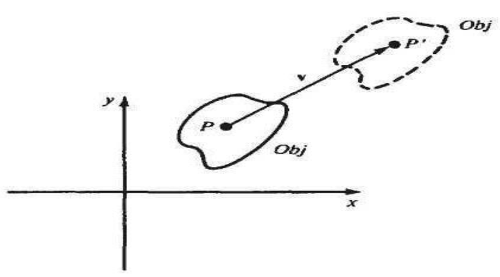


Fig. 4-2

Rotation about the Origin

In rotation, the object is rotated ϴ about the origin. The convention is that the direction of rotation is counterclockwise if 6 is a positive angle and clockwise if 0 is a negative angle (see Fig. 4-3). The transformation of rotation R_θ

p'=R_θ(P)

where x'= $x\cos(\theta)$ -$y\sin(\theta)$ and = $x\sin(\theta)$ +$y\cos(\theta)$

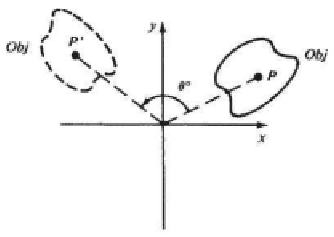


Fig. 4-3

Scaling with Respect to the Origin
Scaling is the process of expanding or compressing the dimensions of an object. Positive scaling constants s_x and s_y are used to describe changes in length with respect to the x direction and y direction, respectively. A scaling constant greater than one indicates an expansion of length, and less than one, compression of length. The scaling fransformation $S_{s_x s_y,}$ is given by P' = $S_{s_x s_y,}$(P) where $x_1 = s_x x$ and $y = s_y y$. Notice that, after a scaling fransformation is performed, the new object is located at a different position relative to the origin. In fact, in a scaling fransformation the only point that remains fixed is the origin. Figure 4-4 shows scaling fransformation with scaling factors $s_x = 2$ and $s_y = \frac{1}{2}$

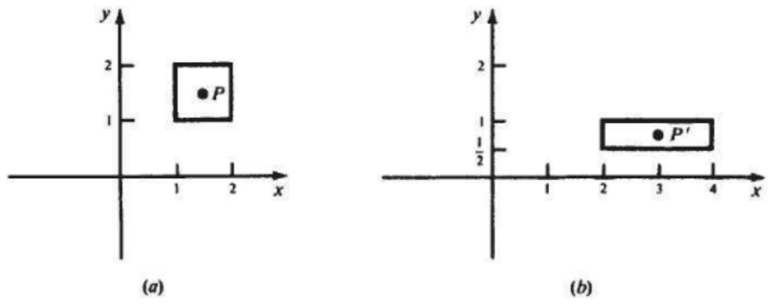


Fig. 4-4

If both scaling constants have the same value s, the scaling fransformation is said to be homogeneous or uniform. Furthermore, if s > 1, it is a magnification and for s < 1, a reduction

<u>Program</u>

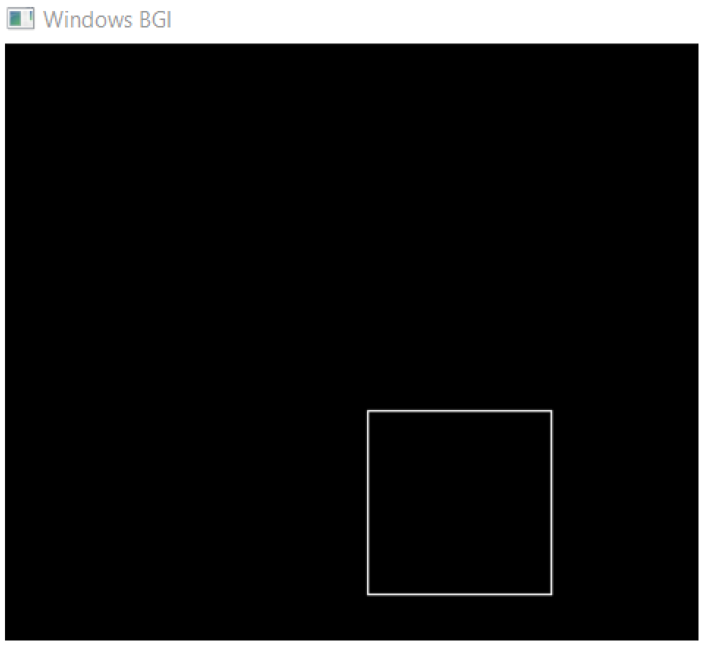
<u>Translate square</u>

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

using namespace std;
int main()
{
   int gd=DETECT,gm;
   initgraph(&gd,&gm,(char*)"");

   int x1=200,y1=200;

   int x2=300,y2=200;

   int x3=200,y3=300;

   int x4=300, y4=300;
```

```
line(x1,y1,x2,y2);
line(x2,y2,x4,y4);
line(x3,y3,x4,y4);
line(x3,y3,x1,y1);

double tx,ty;
cout<<"Enter scaling factor"<<endl;
cin>>tx>>ty;

x1=x1+tx;
y1=y1+ty;

x2=x2+tx;
y2=y2+ty;

x3=x3+tx;
y3=y3+ty;

x4=x4+tx;
y4=y4+ty;
cout<<"x1 "<<x1<<endl;
cout<<"y1 "<<y1<<endl;

cout<<"x2 "<<x2<<endl;
cout<<"y2 "<<y2<<endl;


cout<<"x3 "<<x3<<endl;
cout<<"y3 "<<y3<<endl;

cout<<"x4 "<<x4<<endl;
cout<<"y4 "<<y4<<endl;

cout<<"Squre after Translation \n";
initgraph(&gd,&gm,(char*)"");

line(x1,y1,x2,y2);
line(x2,y2,x4,y4);
 line(x3,y3,x4,y4);
 line(x3,y3,x1,y1);
```

```
    getch();

    closegraph();


}
```

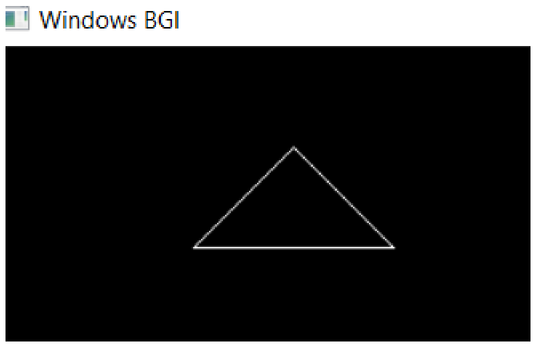
Output



Translate Triangle

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

using namespace std;
int main(){
int gd=DETECT,gm;
initgraph(&gd,&gm,(char*)"");

int x1=100,y1=100;
int x2=200,y2=100;
int x3=150,y3=50;

line(x1,y1,x2,y2);
line(x2,y2,x3,y3);
line(x3,y3,x1,y1);

int x,y;
cout<<"Enter translation factor"<<endl;
cin>>x>>y;

x1+=x;
y1+=y;

x2+=x;
y2+=y;

x3+=x;
y3+=y;
cout<<"x1 "<<x1<<endl;
cout<<"y1 "<<y1<<endl;

cout<<"x2 "<<x2<<endl;
cout<<"y2 "<<y2<<endl;

cout<<"x3 "<<x3<<endl;
cout<<"y3 "<<y3<<endl;
cout<<"Triangle after translation \n";
initgraph(&gd,&gm,(char*)"");
line(x1,y1,x2,y2);
line(x2,y2,x3,y3);
line(x3,y3,x1,y1);


getch();
closegraph();


}
```

Output

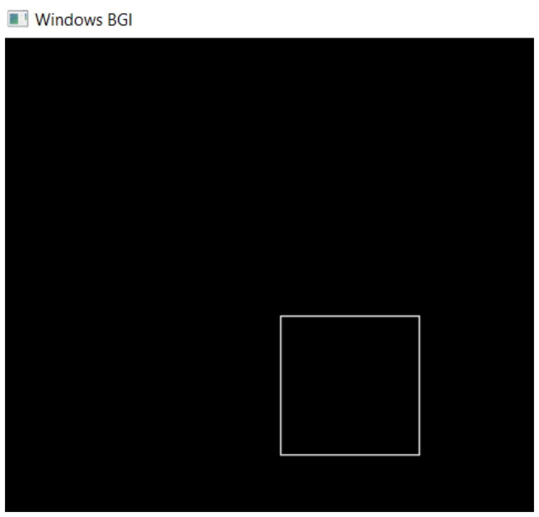


Scale Square

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

using namespace std;
int main()
{
   int gd=DETECT,gm;
   initgraph(&gd,&gm,(char*)"");

   int x1=200,y1=200;
   int x2=300,y2=200;
   int x3=200,y3=300;
   int x4=300, y4=300;

   line(x1,y1,x2,y2);
   line(x2,y2,x4,y4);
   line(x3,y3,x4,y4);
   line(x3,y3,x1,y1);

   double sx,sy;
   cout<<"Enter scaling factor"<<endl;
   cin>>sx>>sy;

   x1=x1*sx;
   y1=y1*sy;
   x2=x2*sx;
```

```
    y2=y2*sy;

    x3=x3*sx;
    y3=y3*sy;

    x4=x4*sx;
    y4=y4*sy;

    cout<<"x1 "<<x1<<endl;
    cout<<"y1 "<<y1<<endl;

    cout<<"x2 "<<x2<<endl;
    cout<<"y2 "<<y2<<endl;


    cout<<"x3 "<<x3<<endl;
    cout<<"y3 "<<y3<<endl;

    cout<<"x4 "<<x4<<endl;
    cout<<"y4 "<<y4<<endl;


    cout<<"Squre after Scalling \n";
    initgraph(&gd,&gm,(char*)"");

    line(x1,y1,x2,y2);
    line(x2,y2,x4,y4);
    line(x3,y3,x4,y4);
    line(x3,y3,x1,y1);

    getch();
    closegraph();

}
```

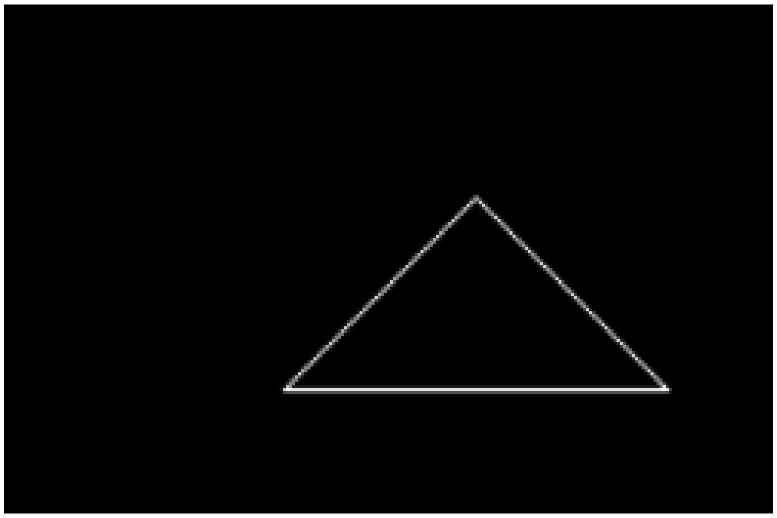
Output



Scale triangle

```
#include<iostream>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
#include<math.h>
using namespace std;
int main(){
int gd=DETECT,gm;
initgraph(&gd,&gm,(char*)"");
int x1=100,y1=100;
int x2=200,y2=100;
int x3=150,y3=50;
line(x1,y1,x2,y2);
line(x2,y2,x3,y3);
line(x3,y3,x1,y1);
double sx,sy;
cout<<"Enter scaling factor"<<endl;
cin>>sx>>sy;
int x4=sx*x1-x1;
int y4=sy*y1-y1;
x1=x1*sx;
y1=y1*sy;
x2=x2*sx;
y2=y2*sy;
x3=x3*sx;
y3=y3*sy;
cout<<"x1 "<<x1<<endl;
cout<<"y1 "<<y1<<endl;
cout<<"x2 "<<x2<<endl;
cout<<"y2 "<<y2<<endl;
cout<<"x3 "<<x3<<endl;
cout<<"y3 "<<y3<<endl;
cout<<"Triangle after rotation \n";
initgraph(&gd,&gm,(char*)"");
line(x1,y1,x2,y2);
line(x2,y2,x3,y3);
line(x3,y3,x1,y1);
getch();
closegraph();

}
```

Output



9. Name Of Program

Write a program to perform line clipping

Theoretical Background

The following algorithms provide efficient ways to decide the spatial relationship between an arbitrary line and the clipping window and to find intersection point(s).

The Cohen-Sutherland Algorithm

In this algorithm we divide the line clipping process into two phases: (1) identify those lines which intersect the clipping window and so need to be clipped and (2) perform the clipping. All lines fall into one of the following *clipping categories:*

1. Visible : both endpoints of the line lie within the window
2. Not visible : the line definitely lies outside the window. This will occur if the line from (xx, yx) to
   (x2,y2) satisfies any one of the following four inequalities:
   x1,x2 > x_{max} y1,y2 > y_{max}
   x1,x2 >< x_{min} y1,y2 < y_{min}
3. Clipping candidate : the line is in neither category 1 nor 2.

The algorithm employs an efficient procedure for finding the category of a line. It proceeds in two steps:

1. Assign a 4-bit region code to each endpoint of the line. The code is determined according to which of the following nine regions of the plane the endpoint lies in

Starting from the leftmost bit, each bit of the code is set to true (1) or false (0) according to the scheme

Bit 1 = endpoint is above the window = sign (y — y_{max})
Bit 2 = endpoint is below the window = sign (y_{min} — y)
Bit 3 = endpoint is to the right of the window = sign (x — x_{max})
Bit 4 = endpoint is to the left of the window = sign (x_{min} — x)

We use the convention that sign(a) = 1 if *a* is positive, 0 otherwise. Of course, a point with code 0000 is inside the window

2. The line is visible if both region codes are 0000, and not visible if the bitwise logical AND of the codes is not 0000, and a candidate for clipping if the bitwise logical AND of the region codes is 0000

For a line in category 3 we proceed to find the intersection point of the line with one of the boundaries of the clipping window, or to be exact, with the infinite extension of one of the boundaries (see Fig. 5-4). We choose an endpoint of the line, say *(xl,yl)*, that is outside the window, i.e., whose region code is not

0000. We then select an extended boundary line by observing that those boundary lines that are candidates for intersection are the ones for which the chosen endpoint must be "pushed across"so as to change a " 1 " in its code to a "0" (see Fig. 5-4). This me e
y = ymax.
If bit 2 is 1, intersect with line y = ymin
If bit 3 is 1, intersect with line x = xmax.
If bit 4 is 1, intersect with line x = xmin.

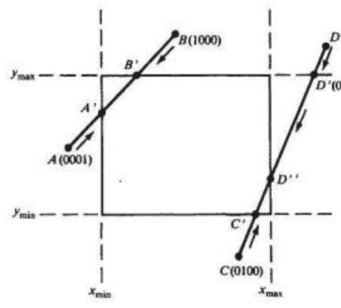


Fig. 5-4

Consider line *CD* in Fig. 5-4. If endpoint *C* is chosen, then the bottom boundary line *y* = *yn* **is** selected for computing intersection. On the other hand, if endpoint *D* is chosen, then either the top boundary line *y* = *ymax* or the right boundary line *x* = xmax is used. The coordinates of the intersection point are

xi=x_{min} or x_{max} if the boundary line is vertical
xi=y1+m(xi-x1)

xi=x1+(yi-y1)/m if the boundary line is horizontal
xi =y_{min} or y_{max}

Program

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

using namespace std;
void storepoints(int,int,int,int,int,int,int[]);
int main()
{
   int gd=DETECT,gm;
   int x1=10,y1=10,x2=100,y2=110;
   //int x1=100,y1=40,x2=100,y2=110;
   int xmax=90,ymax=90,xmin=10,ymin=20;
   int a[10],b[10],flag=0,xi1,yi1,m,xi2,yi2;
```

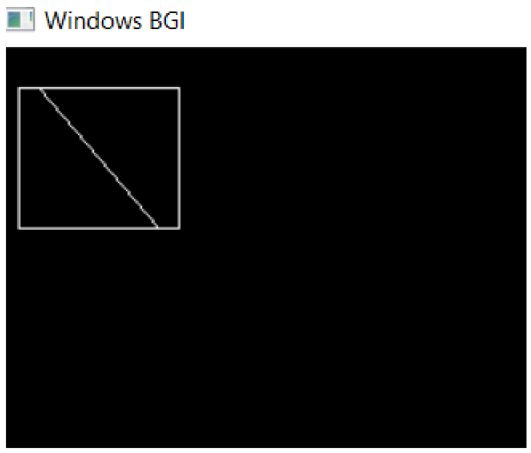
```
storepoints(x2,y2,ymin,ymax,xmax,xmin,b);
storepoints(x1,y1,ymin,ymax,xmax,xmin,a);

for(int i=1; i<=4; i++)
{

   if(a[i]*b[i]==0)
      flag=1;
   else
   {
      flag=0;break;
   }
   cout<<a[1]<<" "<<a[2]<<" "<<a[3]<<" "<<a[4]<<endl;
}

if(flag==1)
{
   m=(y2-y1)/(x2-x1);
   xi1=x1;
   yi1=y1;
}
if(a[1]==1)
{
   yi1=ymax;
   xi1=x1+(1/m)*(yi1-y1);
}
else if(a[2]==1)
{
   yi1=ymin;
   xi1=x1+(1/m)*(yi1-y1);
}

else if(a[3]==1)
{
   xi1=xmax;
   yi1=y1+m*(xi1-x1);
}
else if(a[4]==1)
{
   xi1=xmin;
   yi1=y1+m*(xi1-x1);
}



//for point b
```

```
xi2=x2;
yi2=y2;
if(b[1]==1)
{

   yi2=ymax;
   xi2=x2+(1/m)*(yi2-y2);
}

else if(b[2]==1)
{

   yi2=ymin;
   xi2=x2+(1/m)*(yi2-y2);
}

else if(b[3]==1)
{
   xi2=xmax;
   yi2=y2+m*(xi2-x2);
}
else if(b[4]==1)
{
   xi2=xmin;
   yi2=y2+m*(xi2-x2);
}



initgraph(&gd,&gm,(char*)"");
rectangle(xmin,ymin,xmax,ymax);
line(x1,y1,x2,y2);
delay(5000);
//closegraph();

initgraph(&gd,&gm,(char*)"");
line(xi1,yi1,xi2,yi2);
rectangle(xmin,ymin,xmax,ymax);

if(flag==0)
{
   cout<<"NO Clipping required\n";

}
getch();
```

```
    closegraph();
}
void storepoints(int x1,int y1,int ymin,int ymax,int xmax,int xmin,int c[10])
{
    if((y1-ymax)>0)
        c[1]=1;
    else
        c[1]=0;

    if((ymin-y1)>0)
        c[2]=1;
    else
        c[2]=0;

    if((x1-xmax)>0)
        c[3]=1;
    else
        c[3]=0;

    if((xmin-x1)>0)
        c[4]=1;
    else
        c[4]=0;

    cout<<c[1]<<" "<<c[2]<<" "<<c[3]<<" "<<c[4]<<endl;
}
```

Input & Output

```
1 0 1 0
0 1 0 0
0 1 0 0
0 1 0 0
0 1 0 0
0 1 0 0
```



10. Name Of Program

Write a program to implement reflection of a point, line

Theoretical Background

Mirror Reflection about an Axis

If either the x or y axis is treated as a mirror, the object has a mirror image or reflection. Since the reflection P' of an object point P is located the same distance from the mirror as P (Fig. 4-5), the mirror reflection fransformation Mx about the x axis is given by

$p'=M_x$(P)

where $x' = x$ and $y' = —y$

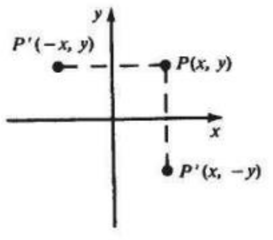


Fig. 4-5

Similarly, the mirror reflection about the *y* axis is

$p'=M_y$(P)

where $x' =- x$ and $y' = y$

Inverse Geometric Transformations

Each geometric transformation has an inverse (see App. 1) which is described by the opposite operation performed by the transformation:

Translation: T_v^{-1} = T_v, or translation in the opposite direction

Rotation: R_θ^{-1} = R_e, or rotation in the opposite direction

Scaling: $S_{Sx,Sy}^{-1}$=$^{S}1/_{s_{x,}}\ 1/_{s_y}$

Mirror reflection: M_x^{-1}*l* = M_x and M_y^{-1}*l* = y

Program

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

```
using namespace std;
int main(){
int gd=DETECT,gm;
initgraph(&gd,&gm,(char*)"");

int x1=100,y1=100;
int x2=200,y2=50;

line(0,240,639,240);
line(320,0,320,479);
line(x1,y1,x2,y2);

int xo1,xo2,yo1,yo2;
int x;
cout<<"Enter 1. for rotation about x axis "<<endl;
cout<<"Enter 2. for rotation about y axis "<<endl;
cout<<"Enter 3. for rotation about both axis "<<endl;
cin>>x;


if(x==1 or x==3)  /// x axis
{
   xo1=640-x1;
   xo2=640-x2;
   yo1=y1;
   yo2=y2;


  line(0,240,639,240);
   line(320,0,320,479);


   line(xo1,yo1,xo2,yo2);
}

if(x==2 or x==3)
{
   yo1=480-y1;
   yo2=480-y2;
   xo1=x1;
```

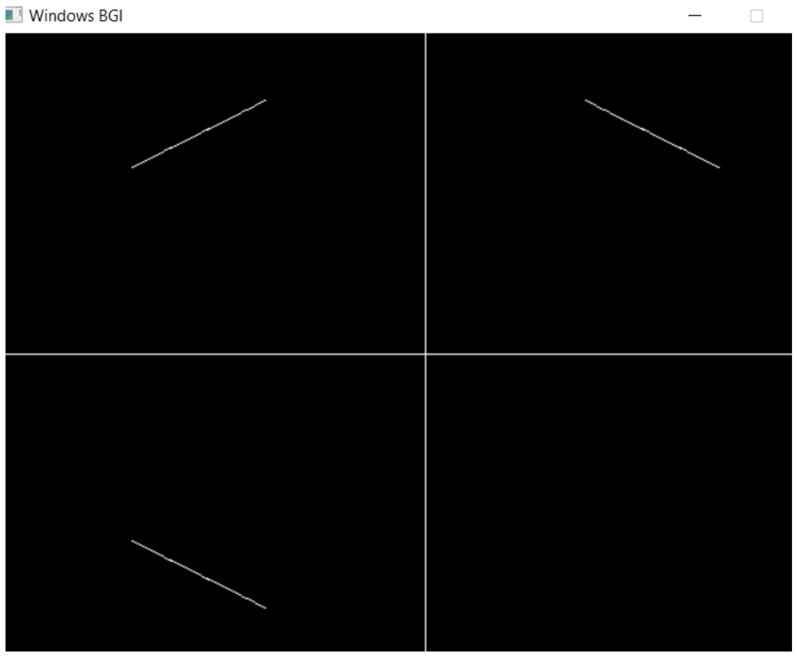
```
    xo2=x2;

    line(0,240,639,240);
    line(320,0,320,479);

    line(xo1,yo1,xo2,yo2);
}


getch();
closegraph();

}
```

Input & Output

Enter 1. for rotation about x axis

Enter 2. for rotation about y axis

Enter 3. for rotation about both axis



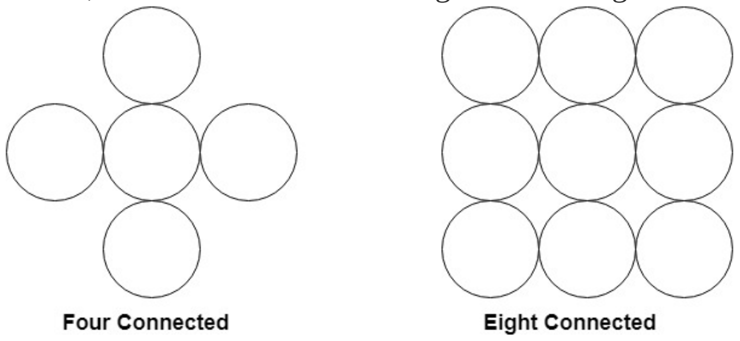
11. Name Of Program

Write a program to implement polygon filling

Theoretical Background

Boundary Filled Algorithm:

This algorithm uses the recursive method. First of all, a starting pixel called as the seed is considered. The algorithm checks boundary pixel or adjacent pixels are colored or not. If the adjacent pixel is already filled or colored then leave it, otherwise fill it. The filling is done using four connected or eight connected approaches.



Four connected approaches is more suitable than the eight connected approaches.

1. Four connected approaches: In this approach, left, right, above, below pixels are tested.

2. Eight connected approaches: In this approach, left, right, above, below and four diagonals are selected.

Boundary can be checked by seeing pixels from left and right first. Then pixels are checked by seeing pixels from top to bottom. The algorithm takes time and memory because some recursive calls are needed.

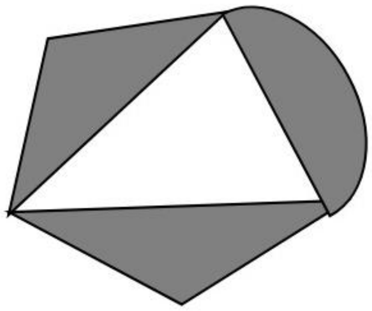
Flood Fill Algorithm:

In this method, a point or seed which is inside region is selected. This point is called a seed point. Then four connected approaches or eight connected approaches is used to fill with specified color.

The flood fill algorithm has many characters similar to boundary fill. But this method is more suitable for filling multiple colors boundary. When boundary is of many colors and interior is to be filled with one color we use this algorithm.

In fill algorithm, we start from a specified interior point (x, y) and reassign all pixel values are currently set to a given interior color with the desired color. Using either a 4-connected or 8-

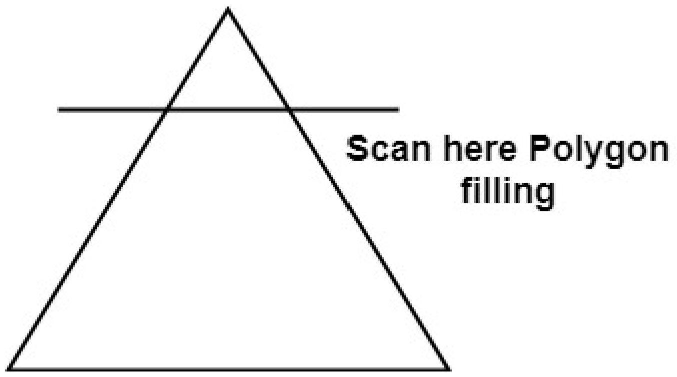
connected approaches, we t   til all interior points have been repainted.



Scan Line Polygon Fill Algorithm:

This algorithm lines interior points of a polygon on the scan line and these points are done on or off according to requirement. The polygon is filled with various colors by coloring various pixels.

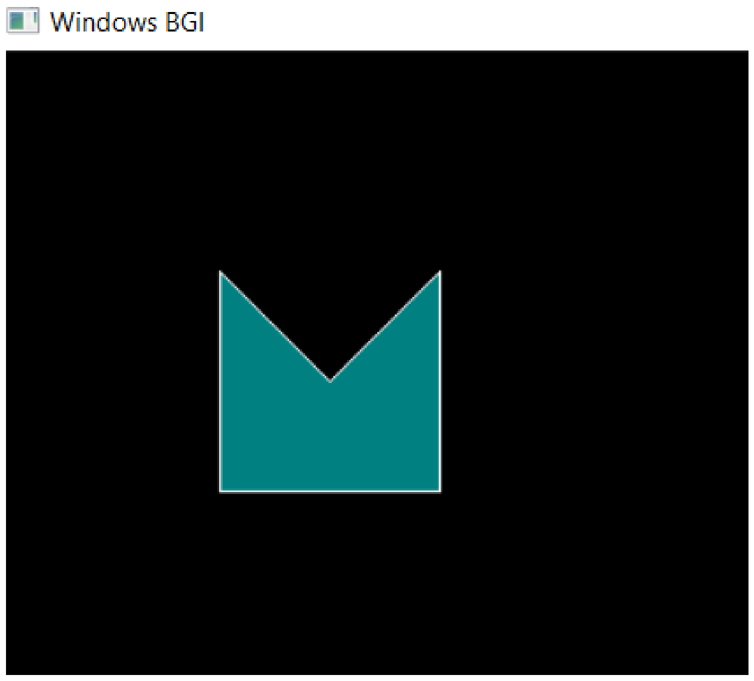
In above figure polygon and a line cutting polygon in shown. First of all, scanning is done. Scanning is done using raster scanning concept on display device. The beam starts scanning from the top left corner of the screen and goes toward the bottom right corner as the endpoint. The algorithms find points of intersection of the line with polygon while moving from left to right and top to bottom. The various points of intersection are stored in the frame buffer. The intensities of such points is keep high. Concept of coherence property is used. According to this property if a pixel is inside the polygon, then its next pixel will be inside the polygon.



Program

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

using namespace std;
int main()
{
    int gd=DETECT,gm;
    initgraph(&gd,&gm,(char*)"");
    int p=1,x;
    int a[12]= {100,100,150,150,200,100,200,200,100,200,100,100};
    drawpoly(6,a);

    //setfillstyle(SOLID_FILL, 3);
   // floodfill(152, 182, WHITE);

    getch();
    closegraph();
}
```

Input & Output



12. **<u>Name Of Program:</u>**

Study of basic graphics functions defined in “graphics.h “.

Theory background:

1) INITGRAPH

- Initializes the graphics system.

Declaration

- Void far initgraph(int far *graphdriver)

Remarks

- To start the graphic system, you must first call initgraph.

- Initgraph initializes the graphic system by loading a graphics driver from disk (or validating a registered driver) then putting the system into graphics mode.

- Initgraph also resets all graphics settings (color, palette, current position, viewport, etc) to their defaults then resets graph.

2) GETPIXEL, PUTPIXEL

- Getpixel gets the color of a specified pixel.

- Putpixel places a pixel at a specified point.

Decleration

- Unsigned far getpixel(int x, int y)

- Void far putpixel(int x, int y, int color)

Remarks

- Getpixel gets the color of the pixel located at (x,y);

- Putpixel plots a point in the color defined at (x, y).

Return value

- Getpixel returns the color of the given pixel.

- Putpixel does not return.

3) CLOSE GRAPH

- Shuts down the graphic system.

Decleration

- Void far closegraph(void);

Remarks

- Close graph deallocates all memory allocated by the graphic system.
- It then restores the screen to the mode it was in before you called initgraph.

Return value

- None.

4) ARC, CIRCLE, PIESLICE

- arc draws a circular arc.
- Circle draws a circle
- Pieslice draws and fills a circular pieslice

Declaration

- Void far arc(int x, int y, int stangle, int endangle, int radius);
- Void far circle(int x, int y, int radius);
- Void far pieslice(int x, int y, int stangle, int endangle, int radius);

Remarks

- Arc draws a circular arc in the current drawing color
- Circle draws a circle in the current drawing color
- Pieslice draws a pieslice in the current drawing color, then fills it using the current fill pattern and fill color.

5) ELLIPSE, FILL ELIPSE, SECTOR

- Ellipse draws an elliptical arc.
- Fill ellipse draws and fills ellipse.
- Sector draws and fills an elliptical pie slice.

Declaration

- Void far ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius)
- Void far fill ellipse(int x, int y, int xradius, int yradius)

 Void farsectoe(int x, int y, int stangle, int endangle, int xradius, int yradius)

Remarks
 Ellipse draws an elliptical arc in the current drawing color.

 Fill ellipse draws an elliptical arc in the current drawing color and than fills it with fill color and fill pattern.

 Sector draws an elliptical pie slice in the current drawing color and than fills it using the pattern and color defined by setfill style or setfill pattern.

6) FLOODFILL
 Flood-fills a bounded region.

Decleration
 Void far floodfill(int x, int y, int border)

Remarks
 Floodfills an enclosed area on bitmap device.

 The area bounded by the color border is flooded with the current fill pattern and fill color.

• (x,y) is a "seed point"
➢ If the seed is within an enclosed area, the inside will be filled.

➢ If the seed is outside the enclosed area, the exterior will be filled.
 Use fillpoly instead of floodfill wherever possible so you can maintain code compatibility with future versions.

 Floodfill doesnot work with the IBM-8514 driver.
Return value
 If an error occurs while flooding a region, graph result returns „1‟.

7) GETCOLOR, SETCOLOR
 Getcolor returns the current drawing color.

 Setcolor returns the current drawing color.

Decleration
 Int far getcolor(void);

 Void far setcolor(int color)

Remarks

 Getcolor returns the current drawing color.

 Setcolor sets the current drawing color to color, which can range from 0 to getmaxcolor.

 To set a drawing color with setcolor , you can pass either the color number or the equivalent color name.

8) LINE,LINEREL,LINETO

 Line draws a line between two specified pints.

 Onerel draws a line relative distance from current position(CP).

 Linrto draws a line from the current position (CP) to(x,y).

Declaration

• Void far lineto(int x, int y)

Remarks

 Line draws a line from (x1, y1) to (x2, y2) using the current color, line style and thickness. It does not update the current position (CP).

 Linerel draws a line from the CP to a point that is relative distance (dx, dy) from the CP, then advances the CP by (dx, dy).

 Lineto draws a line from the CP to (x, y), then moves the CP to (x,y).

Return value

 None

9) RECTANGLE

 Draws a rectangle in graphics mode.

Decleration

 Void far rectangle(int left, int top, int right, int bottom)

Remarks

 It draws a rectangle in the current line style, thickness and drawing color.

 (left, top) is the upper left corner of the rectangle, and (right, bottom) is its lower right corner.

Return value

 None