Idris Ratlamwala

C31 – 2003145

CG - Journal , Assingments & MPR project

SR.No.	TOPICS
1.	Implement DDA Line drawing Method in C
2.	Implement Bresenham's Line drawing Method in C
3.	Implement Midpoint circle drawing Method in C
4.	Implement Midpoint ellipse drawing Method in C
5.	Implement flood fill and boundary fill to fill a polygon.
6.	Implement 2D Transformations on a polygon – Translation, Rotation, Scaling, Reflection and Shear
7.	Implement Bezier curve
8.	Implement Koch Curve for fractal generation
9.	Implement Liang Barsky Line clipping Method in C
10.	Implement Sutherland Hodgeman polygon clipping Method in C
11.	Assingment 1
12.	Assingment 2
13.	Mini Project : [Pacman Game]

Idris Rattamwala 2003145 - C31

Experiment - 1

Aim : Implement DDA Line drawing method in C

Theory:

DDA is an incremental method of scan conversion of lines, stands for ~~Differential~~ Digital Differential Analyzer. In this method calculation is performed at each step but by using previous step's result.

Algorithm:

Steps:

1. Get the two end points (x_0, y_0) & (x_1, y_1) as input.
2. Calculate the difference between them
   ie $dx = x_1 - x_0$
   & $dy = y_1 - y_0$
3. Based on the calculated diffence in ~~pre~~ previous step, we need to identify the number of steps to put pixel ie.
   If $dx > dy$ then we need more steps (i.e. resolution) in x, otherwise y coordinate.

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

4. Calculate the increment in x & y coordinate
   Xinc = dx/dy (float) steps & Yinc = dy / (float) steps.

Basic DDA :

```c
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <math.h>
int sign(float);
void main()
{
  int gd = DETECT, gm, i = 0, k,steps;
  float x1, x2, y1, y2, dx, dy, xinc, yinc, x, y;
  clrscr();
  initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
  printf("Enter start cooordinates:\n");
  scanf("%f %f", &x1, &y1);
  printf("Enter end coordinates:\n");
  scanf("%f %f", &x2, &y2);
  if (x1 == x2 && y1 == y2) // m=45
    putpixel((int)x1, (int)y1, 15);
  else
  {
    dx = x2 - x1;
    dy = y2 - y1;
    if (abs(dx) >= abs(dy)) //m<45
      steps = abs(dx);
    else                    //m>45
      steps = abs(dy);
    xinc = dx / steps;
    yinc = dy / steps;
    x = x1 + 0.5 * sign(xinc);
    y = y1 + 0.5 * sign(yinc);
    while (i < steps)
    {
      putpixel((int)x, (int)y, 15);
      x = x + xinc;
      y = y + yinc;
      i++;
    }
  }
  getch();
  // line(x1,y1,x2,y2);
  closegraph();
}

int sign(float n)
{
  if (n < 0) return -1;
  else if (n > 0) return 1;
  return 0;
}
```

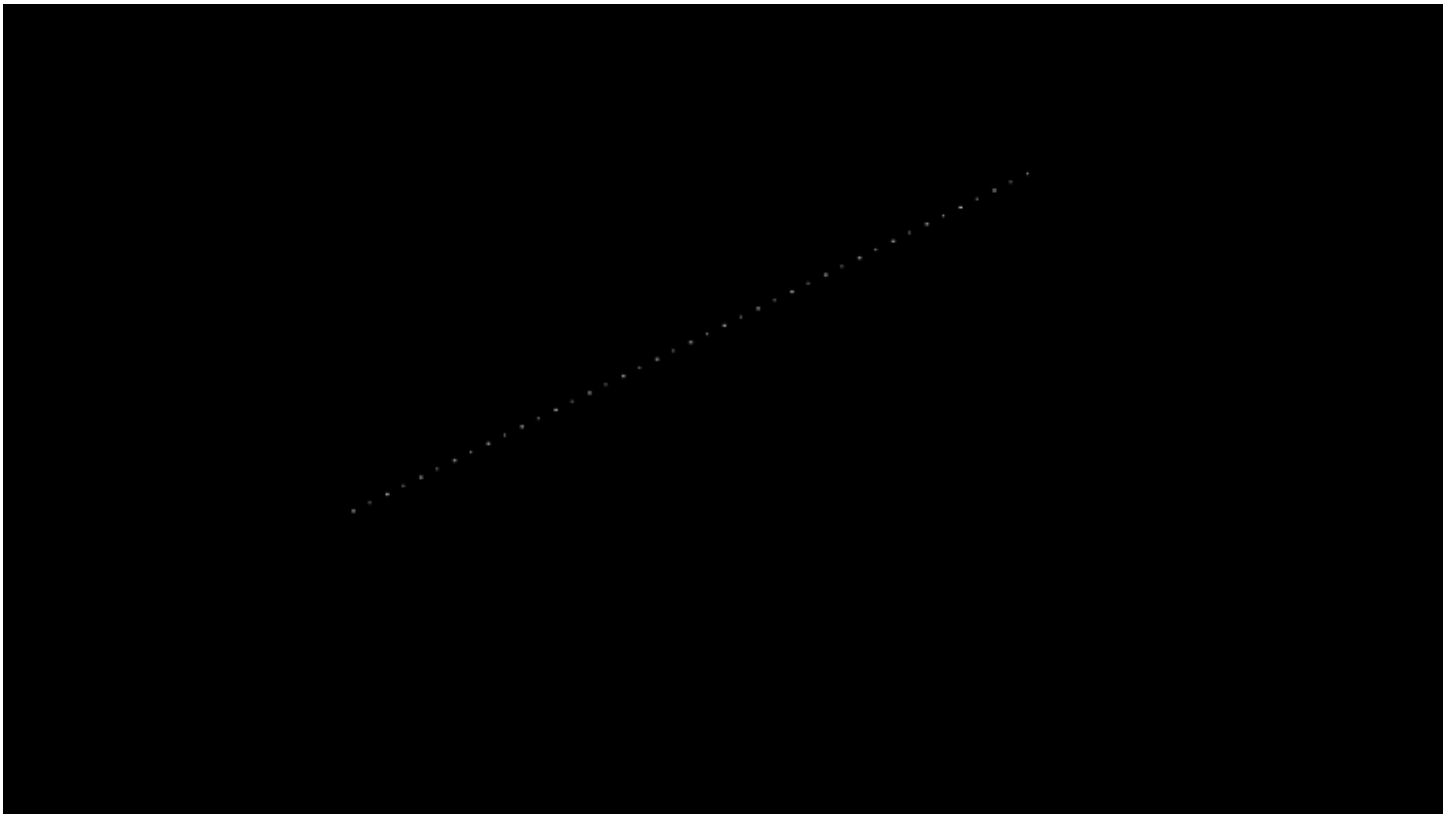
Output :


Enter start cooordinates:
100
200
Enter end coordinates:
500
150

Dotted DDA :

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

int roundNo(float num)
{
    return num < 0 ? num - 0.5 : num + 0.5;
}

void main()
{
    int gd = DETECT, gm = DETECT,dx,dy,steps,k;
    float x1, y1, x2, y2, x, y,xinc,yinc;
    printf("\n Enter two end points of a line:");
    printf("Enter x1,y1: ");
    scanf("%f %f", &x1, &y1);
    printf("Enter x2,y2 : ");
    scanf("%f %f", &x2, &y2);
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    x = x1;
    y = y1;
    putpixel(roundNo(x), roundNo(y), WHITE);
    dx = x2 - x1;
    dy = y2 - y1;
    if (abs(dx) > abs(dy))
        steps = abs(dx);
    else
        steps = abs(dy);
    xinc = (float)dx / steps;
    yinc = (float)dy / steps;
    for (k = 1; k <= steps; k++)
    {
        x = x + xinc;
        y = y + yinc;
        if ((int)roundNo(x) % 10 == 0 || (int)roundNo(y) % 10 == 0)
            putpixel(roundNo(x), roundNo(y), WHITE);
        else
            putpixel(roundNo(x), roundNo(y), BLACK);
    }
    getch();
}
```

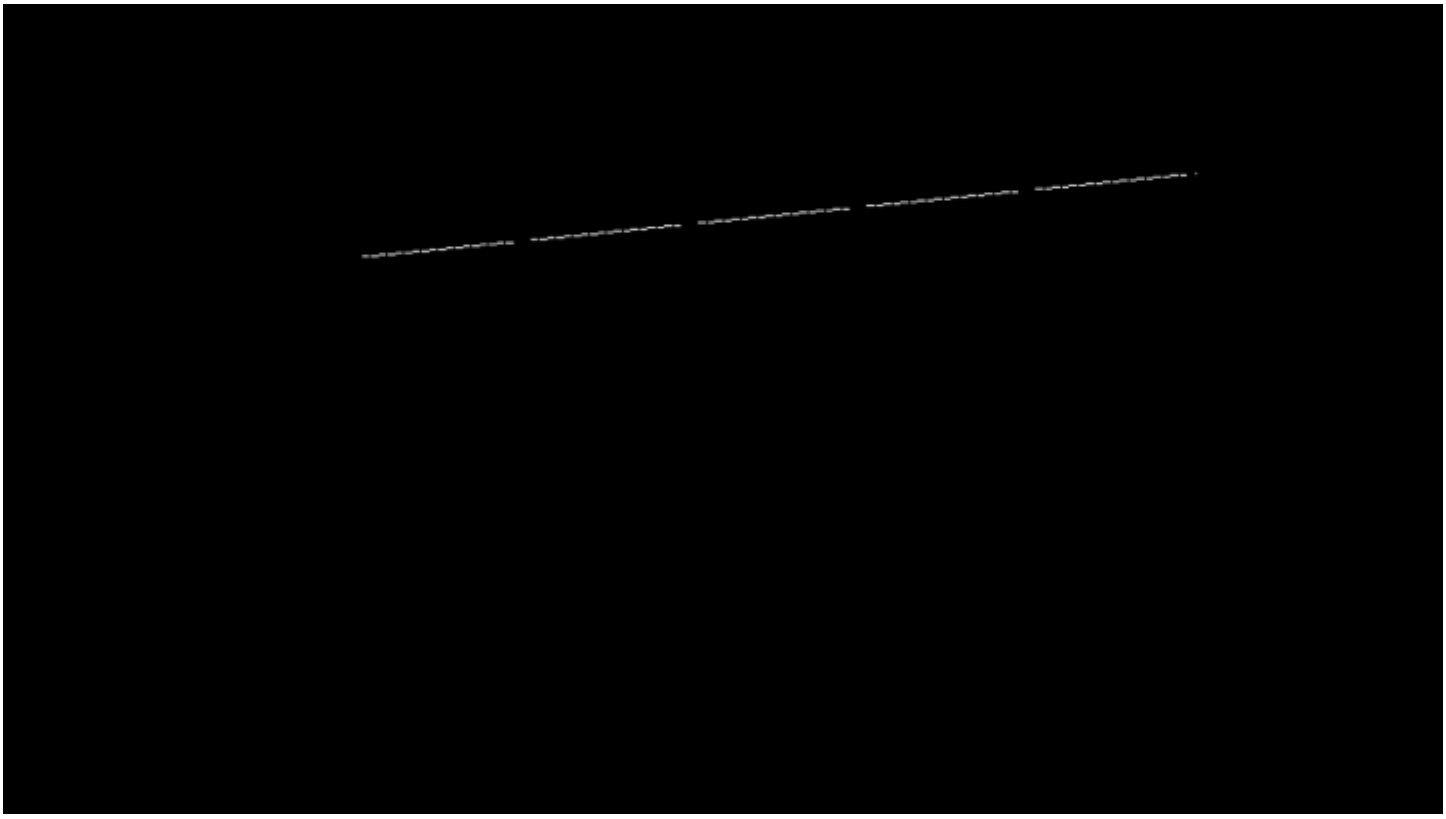
```
Enter start cooordinates:
100
200
Enter end coordinates:
500
150
```



Dashed DDA :

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

int roundNo(float num)
{
    return num < 0 ? num - 0.5 : num + 0.5;
}

void main()
{
    int gd = DETECT, gm = DETECT,dx,dy,steps,k;
    float x1, y1, x2, y2, x, y,xinc,yinc;
    printf("\n Enter two end points of a line:");
    printf("Enter x1,y1: ");
    scanf("%f %f", &x1, &y1);
    printf("Enter x2,y2 : ");
    scanf("%f %f", &x2, &y2);
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    x = x1;
    y = y1;
    putpixel(roundNo(x), roundNo(y), WHITE);
    dx = x2 - x1;
    dy = y2 - y1;
    if (abs(dx) > abs(dy))
        steps = abs(dx);
    else
        steps = abs(dy);
    xinc = (float)dx / steps;
    yinc = (float)dy / steps;
    for (k = 1; k <= steps; k++)
    {
        x = x + xinc;
        y = y + yinc;
        if ((int)roundNo(x) % 10 == 0 || (int)roundNo(y) % 10 == 0)
            putpixel(roundNo(x), roundNo(y), BLACK);
        else
            putpixel(roundNo(x), roundNo(y), WHITE);
    }
    getch();
}
```

```
 Enter two end points of a line:Enter x1,y1: 600 100
Enter x2,y2 : 100 150
```



2003145

PAGE NO. 6m²
DATE / /

Experiment - 2

Aim: Implement Bresenham Line drawing method in C.

Theory:

Bresenham's line drawing algorithm that determines the points of an n-dimensional raster that should be selected in order to form a close approximation to a straight line between two points. It is commonly used to form a draw lines on a computer screen, as it uses only integer addition / subtraction and bit shifting all of which are very cheap operations in standard computer architectures.

It was one of the earliest algorithm developed in the field of computer graphics. An extension to the original algorithm may be used for drawing circles.

Algorithm:

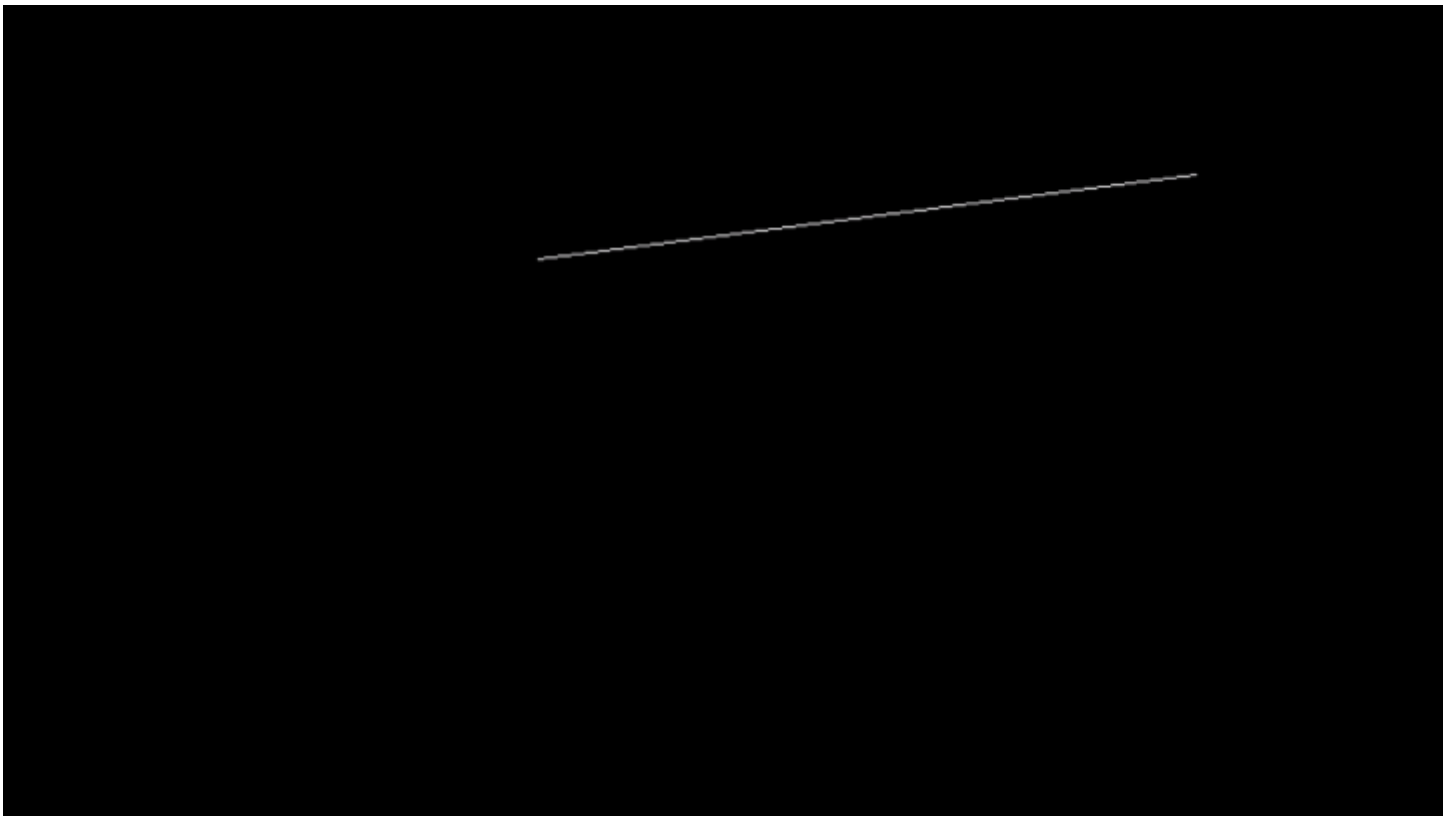
Steps:

1) Get the two end points (x_0, y_0) as left end point and (x_1, y_1) as right end point.
2) Calculate values of dx, dy, $2dy$ & $2dy - dx$.
3) Put pixel (x_0, y_0)
4) Let $P_0 = 2dy - dx$
5) At each x_k along the line (starting with $k=0$);
   if, $P_k < 0$, then next point to plot is (x_{k+1}, y_k);
   And $P_{k+1} = P_k + 2dy$;
   else, point to plot is (x_{k+1}, y_{k+1});
   & $P_{k+1} = P_k + 2dy - 2dx$;
6) Repeat step 4 dx times.

Bresanham basic :

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

int roundNo(float num)
{
    return num < 0 ? num - 0.5 : num + 0.5;
}

void main()
{
    int gd = DETECT, gm, i, dx, dy, s1, s2, f = 0, p;
    float x1, y1, x2, y2, x, y, temp;
    clrscr();

    printf("Enter x1,y1: ");
    scanf("%f %f", &x1, &y1);
    printf("Enter x2,y2 : ");
    scanf("%f %f", &x2, &y2);
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    x = x1;
    y = y1;
    putpixel(roundNo(x), roundNo(y), WHITE);
    dx = abs(x2 - x1);
    dy = abs(y2 - y1);
    if (x2 - x1 < 0)
    s1 = -1;
    if (x2 - x1 > 0)
    s1 = 1;
    if (y2 - y1 < 0)
    s2 = -1;
    if (y2 - y1 > 0)
    s2 = 1;
    if (dy > dx)
    {
    temp = dx;
    dx = dy;
    dy = temp;
    f = 1;
    }
    p = 2 * dy - dx;
    for (i = 1; i <= dx; i++)
    {
    if (p < 0)
    {
        if (f == 1)
        y = y + s2;
```

```
        else
        x = x + s1;
        p = p + 2 * dy;
    }
    else
    {
        x = x + s1;
        y = y + s2;
        p = p + 2 * dy - 2 * dx;
    }
    putpixel(roundNo(x), roundNo(y), WHITE);
    }
    getch();
}
```

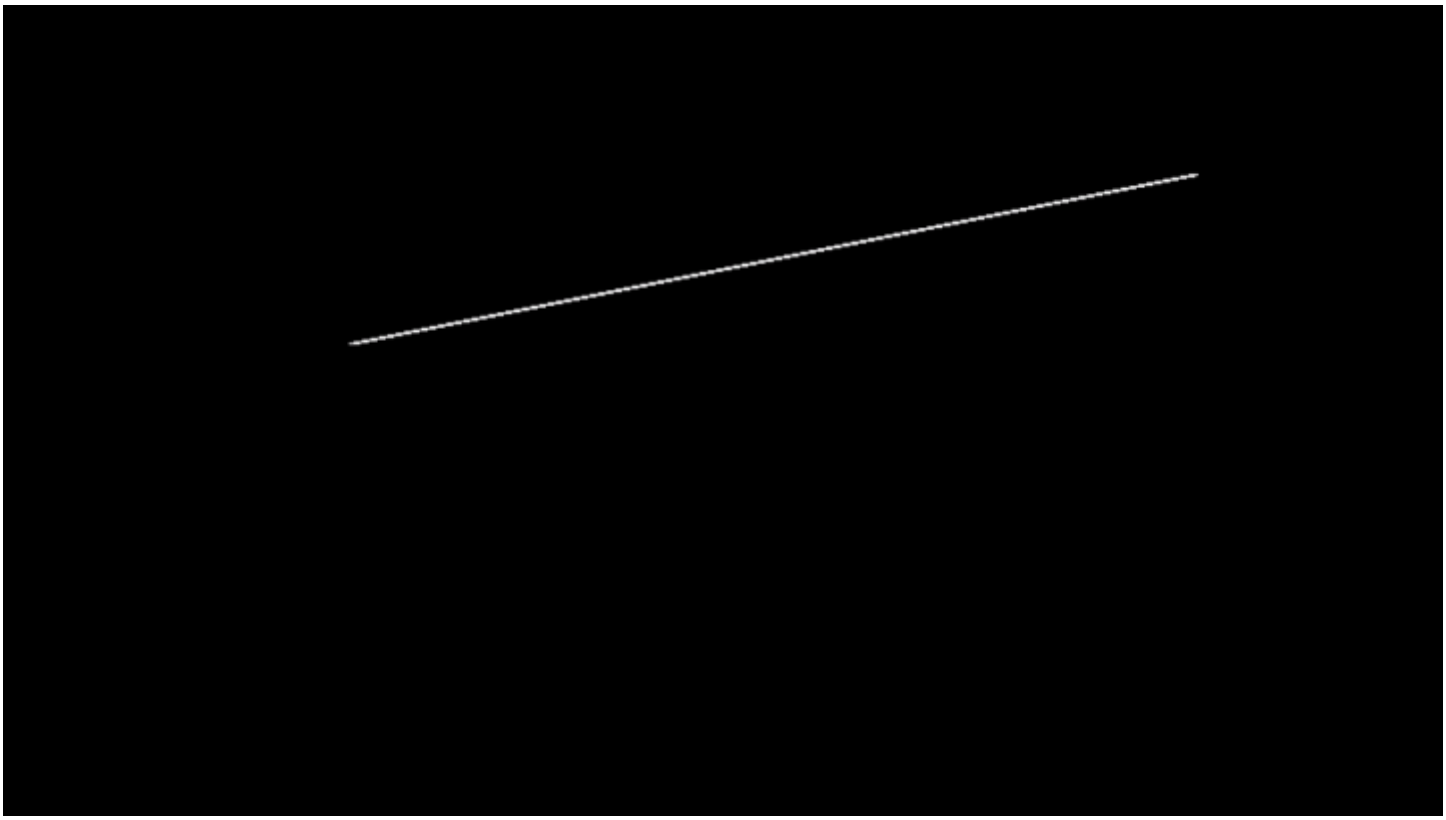


Bresanham thick :

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

int roundNo(float num)
{
    return num < 0 ? num - 0.5 : num + 0.5;
}

 void main()
{
 int gd = DETECT, gm = DETECT;
 float x1, y1, x2, y2, x, y, temp;
 int s1, s2, f = 0, p, choice,dx,dy,i;
 printf("\n Enter two end points of a line:");
 printf("Enter x1,y1: ");
 scanf("%f %f", &x1, &y1);
 printf("Enter x2,y2 : ");
 scanf("%f %f", &x2, &y2);
 printf("Enter thickness:1,2,3,4,5:");
 scanf("%d", &choice);
initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
 x = x1;
 y = y1;
 putpixel(roundNo(x), roundNo(y), WHITE);
  dx = abs(x2 - x1);
  dy = abs(y2 - y1);
 if (x2 - x1 < 0)
 s1 = -1;
 if (x2 - x1 > 0)
 s1 = 1;
 if (y2 - y1 < 0)
s2 = -1;
 if (y2 - y1 > 0)
 s2 = 1;
 if (dy > dx)
 {
 temp = dx;
 dx = dy;
 dy = temp;
 f = 1;
 }
 p = 2 *dy - dx;
 for (i = 1; i <= dx; i++)
 {
 if (p < 0)
 {
```

```
 if (f == 1)
 y = y + s2;
 else
 x = x + s1;
 p = p + 2 * dy;
 }
 else
 {
 x = x + s1;
 y = y + s2;
 p = p + 2 *dy - 2 * dx;
 }
 switch (choice)
 {
 case 1:
 putpixel(roundNo(x), roundNo(y), WHITE);
 break;
 case 2:
 putpixel(roundNo(x), roundNo(y), WHITE);
 putpixel(roundNo(x + 1), roundNo(y), WHITE);
 break;
 case 3:
putpixel(roundNo(x), roundNo(y), WHITE);
 putpixel(roundNo(x + 1), roundNo(y), WHITE);
 putpixel(roundNo(x - 1), roundNo(y), WHITE);
 break;
 case 4:
 putpixel(roundNo(x), roundNo(y), WHITE);
 putpixel(roundNo(x + 1), roundNo(y), WHITE);
 putpixel(roundNo(x - 1), roundNo(y), WHITE);
 putpixel(roundNo(x + 2), roundNo(y), WHITE);
 break;
 case 5:
 putpixel(roundNo(x), roundNo(y), WHITE);
 putpixel(roundNo(x + 1), roundNo(y), WHITE);
 putpixel(roundNo(x - 1), roundNo(y), WHITE);
 putpixel(roundNo(x + 2), roundNo(y), WHITE);
 putpixel(roundNo(x - 2), roundNo(y), WHITE);
 break;
 default:
 printf("Wrong input");
 }
 }
 getch();
}
```

```
 Enter two end points of a line:Enter x1,y1: 600 100
Enter x2,y2 : 100 200
Enter thickness:1,2,3,4,5:5
```



2003143

Experiment - 3

Aim: Implement midpoint circle drawing method in C.

Theory:

In computer graphics, the midpoint circle drawing algorithm is used to calculate all the perimeter point, of a circle. In this algorithm, the midpoint between two pixel is calculated which helps in calculating decision parameter.

The value of decision parameter will decide which pixel should be choosen for drawing the circle.

This algorithm only calculates the points for one octant and the points for other octant are generated using 8-way symmetry of circle.

Algorithm:

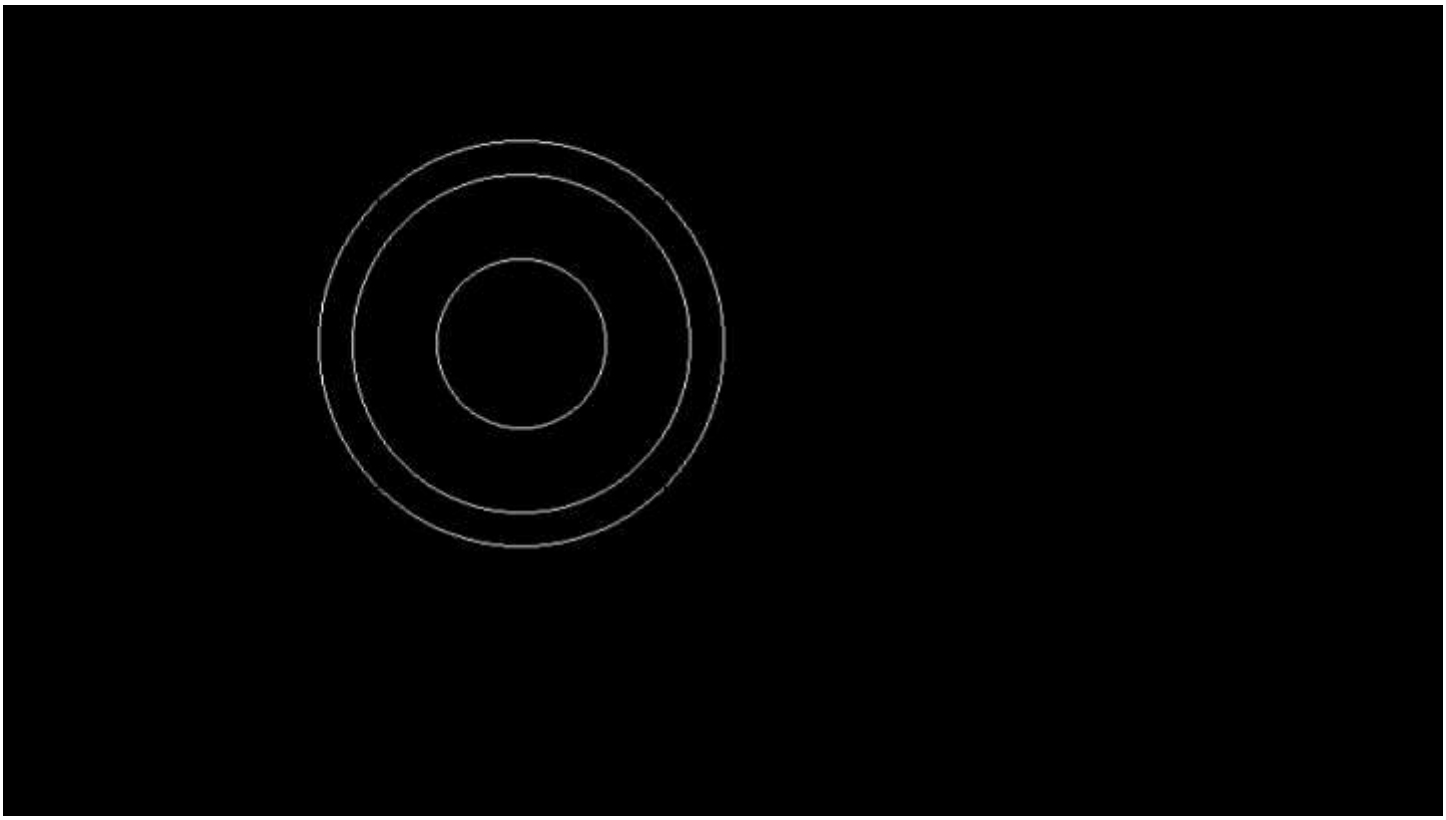
Steps,

1. Get the radius and coordinate of centre from user.
2. Find the decision parameter that decides the nearest point to select. using $d = 5/4 - r$.
3. While $y > x$, do (step 3 & 4)
   if $d < 0$, then
   $y = y,\ x = x + 1,\ d = 2x + 1$.
   else
   $y = y - 1,\ x = x + 1,\ d = d + 2x + 1 + d - 2y$
4. Determine and plot the symmetric points for all the 8 octants.

Midpoint method with cocentric circles :

```c
#include <stdio.h>
#include <graphics.h>
void drawCircle(int xc, int yc, int x, int y)
{
    putpixel(xc + x, yc + y, WHITE);
    putpixel(xc - x, yc + y, WHITE);
    putpixel(xc + x, yc - y, WHITE);
    putpixel(xc - x, yc - y, WHITE);
    putpixel(xc + y, yc + x, WHITE);
    putpixel(xc - y, yc + x, WHITE);
    putpixel(xc + y, yc - x, WHITE);
    putpixel(xc - y, yc - x, WHITE);
}
int main()
{
    int gd = DETECT, gm = DETECT, i, x, y, p;
    int xc, yc, rad[20], n;
    printf("Enter co-ordinates of centre(x,y): \n");
    scanf("%d %d", &xc, &yc);
    printf("Enter the number of cricles:\n");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("\nEnter Radius for circle %d: ", i + 1);
        scanf("%d", &rad[i]);
    }
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    for (i = 0; i < n; i++)
    {
        x = 0;
        y = rad[i];
        p = 1 - rad[i];
        do
        {
            drawCircle(xc, yc, x, y);
            if (p < 0)
                p = p + 2 * x + 3;
            else
            {
                p = p + 2 * (x - y) + 5;
                y--;
            }
            x++;
        } while (x < y);
    }
    getch();
    return (0);
}
```

```
Enter co-ordinates of centre(x,y):
200 200
Enter the number of cricles:
3

Enter Radius for circle 1: 50

Enter Radius for circle 2: 100

Enter Radius for circle 3: 120_
```



2003143

PAGE NO. Ishita

DATE / /

Experiment - 4

Aim: Implement midpoint ellipse drawing method in C.

Theory: In computer graphics, the mid point ellipse algorithm is an incremental method of drawing an ellipse. It is very similar to the mid point algorithm used in the generation of circle.

The mid point algorithm is used to calculate all the perimeter points of an ellipse. In this algorithm, the mid point between 2 pixels is calculated which helps in calculating the decision parameter. The the value of decision parameter determines whether the mid point lies inside, outside or on the ellipse bending. and then position of mid point helps in drawing the ellipse.

Algorithm:

1) Input rx, ry and centre of ellipse (x_c, y_c) And obtain the first point on an ellipse centred on the origin as

$$(x_0, y_0) = (0, r_y)$$

2) Calculate the initial parameter in region 1 as $p1_0 = r_y^2 + r_x^2 r_y + \frac{1}{4} r^2$

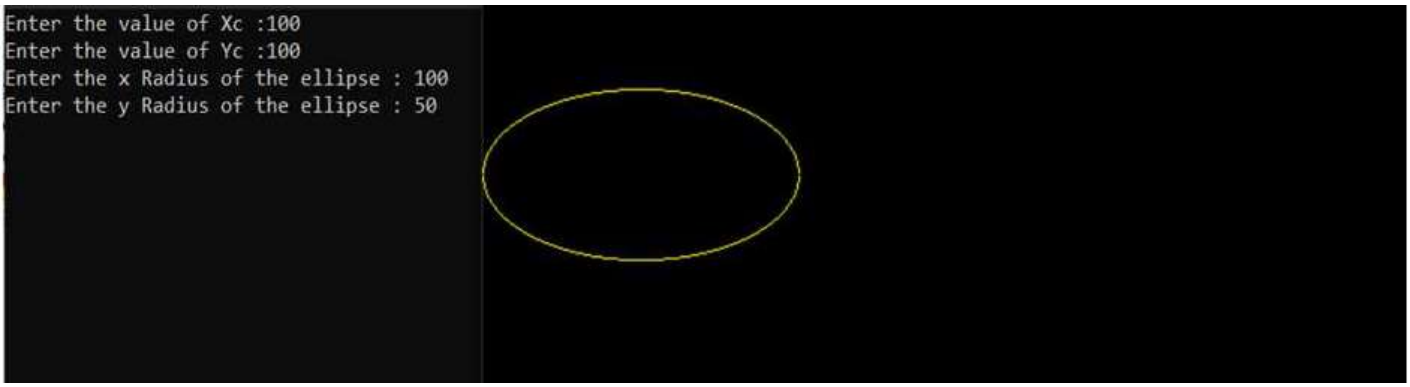
3) At each x_i position start at $i = 0$,
if $p1_i < 0$, the next point is along ellipse centre on $(0, 0)$ is $(x_i + 1, y_i)$
and $p1_{i+1} = p1_i + 2r_y^2 x_{i+1} + r_y^2$

Program :

```
#include <stdio.h>
#include <graphics.h>
#include <math.h>
int main()
{
    int d1, d2;
    int gd = DETECT, gm , x, y;
    int xc, yc, rx, ry, rxsq, rysq, tworxsq, tworysq, dx, dy;
    printf("Enter the value of Xc :");
    scanf("%d", &xc);
    printf("Enter the value of Yc :");
    scanf("%d", &yc);
    printf("Enter the x Radius of the ellipse : ");
    scanf("%d", &rx);
    printf("Enter the y Radius of the ellipse : ");
    scanf("%d", &ry);
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    rxsq = rx * rx;
    rysq = ry * ry;
    tworxsq = 2 * rxsq;
    tworysq = 2 * rysq;
    x = 0;
    y = ry;
    d1 = rysq - (rxsq * ry) + (0.25 * rxsq);
    dx = tworysq * x;
    dy = tworxsq * y;
    do
    {
        putpixel(xc + x, yc + y, YELLOW);
        putpixel(xc - x, yc - y, YELLOW);
        putpixel(xc + x, yc - y, YELLOW);
        putpixel(xc - x, yc + y, YELLOW);
        if (d1 < 0)
        {
            x = x + 1;
            y = y;
            dx = dx + tworysq;
            d1 = d1 + dx + rysq;
        }
        else
        {
            x = x + 1;
            y = y - 1;
            dx = dx + tworysq;
            dy = dy - tworxsq;
```

```
            d1 = d1 + dx - dy + rysq;
        }
    } while (dx < dy);
    d2 = rysq * (x + 0.5) * (x + 0.5) + rxsq * (y - 1) * (y - 1) - rxsq * rysq;
    do
    {
        putpixel(xc + x, yc + y, YELLOW);
        putpixel(xc - x, yc - y, YELLOW);
        putpixel(xc + x, yc - y, YELLOW);
        putpixel(xc - x, yc + y, YELLOW);
        if (d2 > 0)
        {
            x = x;
            y = y - 1;
            dy = dy - tworxsq;
            d2 = d2 - dy + rxsq;
        }
        else
        {
            x = x + 1;
            y = y - 1;
            dy = dy - tworxsq;
            dx = dx + tworysq;
            d2 = d2 + dx - dy + rxsq;
        }
    } while (y > 0);
    getch();
    return (0);
}
```

Output :



2003145

PAGE No.
DATE / /

Experiment-5

Aim: Implement flood fill and boundary Fill method to fill a polygon.

Theory:

Flood fill Algorithm:

In this algorithm, 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 colour.

The Flood fill algorithm has many characteristics similar to boundary fill method. But this method is more suitable for filling multiple colours boundary.

Boundary fill Algorithm:

This algorithm is used frequently in CG to fill a desired colour inside a closed polygon having same boundary colour for all of its sides.

The most appropriate algorithm of the implementation is a stack based recursion function

It follows an approach where region filling begins from some extent residing inside the region and paint inside towards the boundary. It mainly implemented within interactive painting packages, where inside points are easily chosen.

Flood fill

Program :

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

void flood(int x, int y, int newcolor, int oldcolor)
{
    if (getpixel(x, y) == oldcolor)
    {
        putpixel(x, y, newcolor);
        flood(x + 1, y, newcolor, oldcolor);
        flood(x, y + 1, newcolor, oldcolor);
        flood(x - 1, y, newcolor, oldcolor);
        flood(x, y - 1, newcolor, oldcolor);
    }
    delay(2);
}

int main()
{
    int gm, gd = DETECT, radius;
    int x, y, l, b;
    clrscr();
    initgraph(&gd, &gm, "c:\\Turboc3\\bgi");
    printf("Enter the coordinates of the first vertice: \n");
    printf("x: ");
    scanf("%d", &x);
    printf("y: ");
    scanf("%d", &y);
    printf("Enter the length and the breadth of the rectangle: \n");
    printf("length: ");
    scanf("%d", &l);
    printf("breadth: ");
    scanf("%d", &b);
    rectangle(x, y, x + b, y + l);
    flood(x + 1, y + 1, YELLOW, 0);
    closegraph();
    getch();
    return 0;
}
```

Output :

Boundary Fill :

Program :

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

void boundaryFill8(int x, int y, int fill_color, int boundary_color)

{

    if (getpixel(x, y) != boundary_color &&
        getpixel(x, y) != fill_color)
    {
        putpixel(x, y, fill_color);

        boundaryFill8(x + 1, y, fill_color, boundary_color);
        boundaryFill8(x, y + 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y, fill_color, boundary_color);
        boundaryFill8(x, y - 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y - 1, fill_color, boundary_color);
        boundaryFill8(x - 1, y + 1, fill_color, boundary_color);
        boundaryFill8(x + 1, y - 1, fill_color, boundary_color);
        boundaryFill8(x + 1, y + 1, fill_color, boundary_color);
```

```
    }
}

void main()

{
    int gd = DETECT, gm, x, y, l, b;
    clrscr();
    initgraph(&gd, &gm, "c:\\Turboc3\\bgi");
    printf("Enter the coordinates of the first vertice: \n");
    printf("x: ");
    scanf("%d", &x);
    printf("y: ");
    scanf("%d", &y);
    printf("Enter the length and the breadth of the rectangle: \n");
    printf("length: ");
    scanf("%d", &l);
    printf("breadth: ");
    scanf("%d", &b);
    rectangle(x, y, x + b, y + l);
    boundaryFill8(x + 1, y + 1, YELLOW, 15);
    getch();
    closegraph();
}
```

Output :

2053143

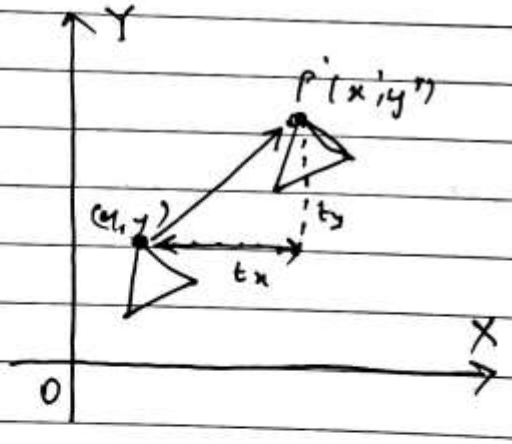
PAGE No.
DATE / /

Expt. 6

Aim: Implement 2D transformation on a polygon
- Translation
- Rotation
- Scaling
- Reflection
- Shearer

Algorithm:

Translation: Shifting of an object to a different position on the screen is called translation.
We can translate a point in 2D by adding translation coordinates (t_x, t_y) to original coordinates.



$x' = x + t_x$

$y' = y + t_y$

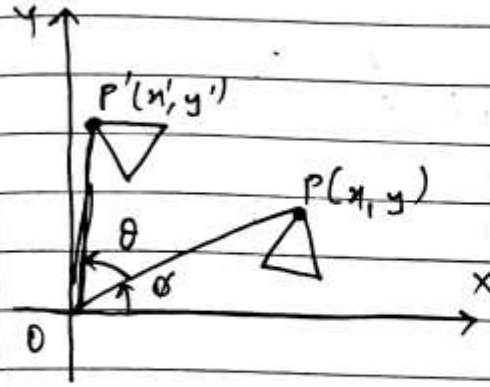
Representation using column vector:

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

& $$T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$ ie $P' = P + T$

DATE / /

Rotation: The object is rotated about a particular angle point at particular angle. θ (theta)



w.k.t,

$x = r\cos\phi$ ——(1)

$y = r\sin\phi$ ——(2)

$\therefore x' = r\cos(\theta+\phi) = r\cos\theta\cos\phi - r\sin\theta\sin\phi$ ——(3)

$\therefore y' = r\sin(\theta+\phi) = r\sin\theta\cos\phi + r\cos\theta\sin\phi$ ——(4)

Substituting 1 in 3 & 2 in 4,

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

$\therefore y' = x\sin\theta + y\cos\theta$.

ie. $P' = P.R$

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

Scaling: In scaling the size of object is changed.

We can either compress or expand the dimensions of object.

Let scaling factor be S_x, S_y

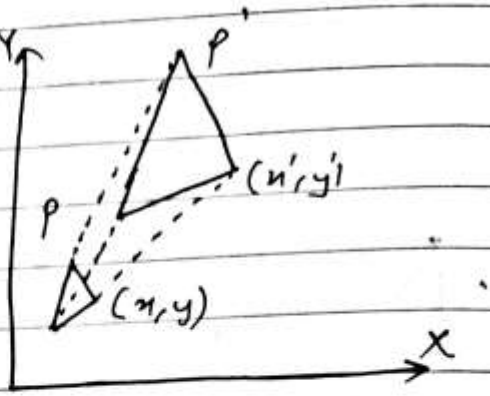
then,

$x' = x \cdot sx$

& $y' = y \cdot sy$

or $P' = P \cdot S$

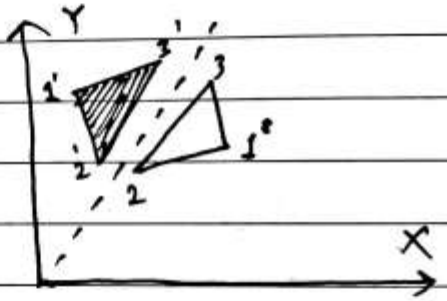
where $S = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix}$



Reflection :

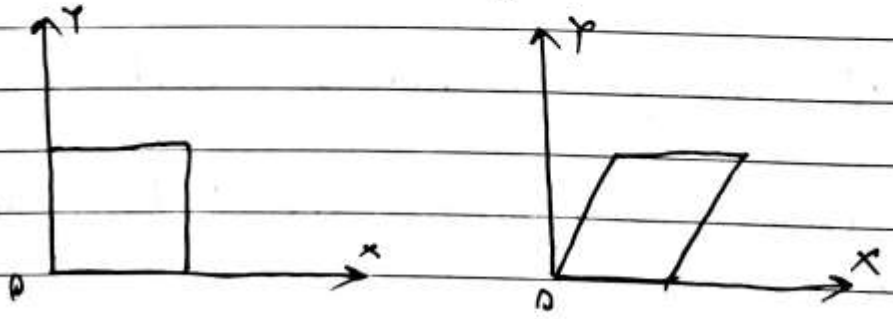
The mirror image of object with respect to an axis is called reflection.

In this transformation size of object doesn't change



Shear :

A transformation which alters the shape of object is called the shearing transformation.



original obj

after x - shear

Program :

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <graphics.h>
int options()
{
    int choice;
 printf("\n1.Translation\n2.Scaling\n3.Rotation\n4.Reflection\n5.
Shearing");
 printf("\nEnter your choice [1 - 5]: ");
 scanf("%d",&choice);
 return choice;
}
void draw(float D[3][3])
{
    line(320 + D[0][0], 240 - D[0][1], 320 + D[1][0], 240 - D[1][1]);
    line(320 + D[1][0], 240 - D[1][1], 320 + D[2][0], 240 - D[2][1]);
    line(320 + D[2][0], 240 - D[2][1], 320 + D[0][0], 240 - D[0][1]);
}
void print(float A[3][3])
{
    int i, j;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            printf("%0.2f\t", A[i][j]);
        }
        printf("\n");
    }
}
void multiply(float B[3][3], float C[3][3], float D[3][3])
{
    int i, j, k;
    float s = 0.0;
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
        {
            s = 0.0;
            for (k = 0; k < 3; k++)
            {
                s = s + (B[i][k] * C[k][j]);
            }
```

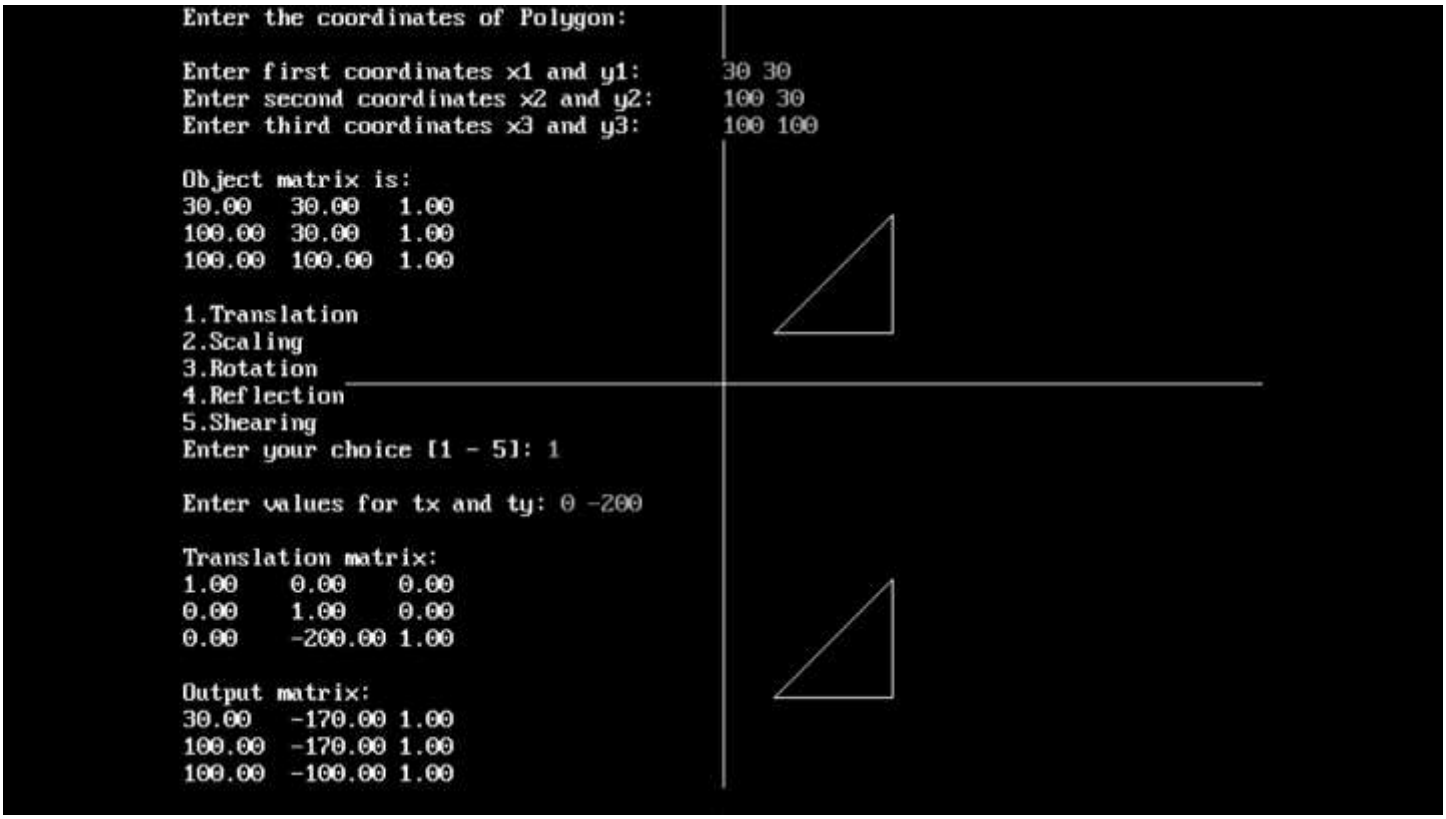
```
            D[i][j] = s;
        }
    }
}
void main()
{
    int gd = DETECT, gm;
    int i, j, k, choice;
    float x1, y1, x2, y2, x3, y3, obj[3][3], T[3][3], S[3][3], R[3][3], Sh[3][3], r[3][3];
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    line(0, 240, 640, 240);
    line(320, 0, 320, 480);
    printf("\nEnter the coordinates of Polygon:\n");
    printf("\nEnter first coordinates x1 and y1: \t");
    scanf("%f%f", &x1, &y1);
    printf("Enter second coordinates x2 and y2:\t");
    scanf("%f%f", &x2, &y2);
    printf("Enter third coordinates x3 and y3: \t");
    scanf("%f%f", &x3, &y3);
    line(320 + x1, 240 - y1, 320 + x2, 240 - y2);
    line(320 + x2, 240 - y2, 320 + x3, 240 - y3);
    line(320 + x3, 240 - y3, 320 + x1, 240 - y1);
    obj[0][0] = x1;
    obj[0][1] = y1;
    obj[1][0] = x2;
    obj[1][1] = y2;
    obj[2][0] = x3;
    obj[2][1] = y3;
    obj[0][2] = 1;
    obj[1][2] = 1;
    obj[2][2] = 1;
    printf("\nObject matrix is: \n");
    print(obj);
    switch (options())
    {
    case 1:
    {
        float tx, ty, trans[3][3];
        printf("\nEnter values for tx and ty: ");
        scanf("%f%f", &tx, &ty);
        for (i = 0; i < 3; i++)
        {
            for (j = 0; j < 3; j++)
            {
                if (i == j)
                    trans[i][j] = 1;
                else
                    trans[i][j] = 0;
            }
        }
```

```
        trans[2][0] = tx;
        trans[2][1] = ty;
        printf("\nTranslation matrix: \n");
        print(trans);
        multiply(obj, trans, T);
        printf("\nOutput matrix: \n");
        print(T);
        draw(T);
    }
    break;
    case 2:
    {
        float scale[3][3], sx, sy;
        printf("\nEnter values for sx and sy: ");
        scanf("%f%f", &sx, &sy);
        for (i = 0; i < 3; i++)
        {
            for (j = 0; j < 3; j++)
            {
                scale[i][j] = 0;
            }
        }
        scale[0][0] = sx;
        scale[1][1] = sy;
        scale[2][2] = 1;
        printf("\nScaling matrix: \n");
        print(scale);
        multiply(obj, scale, S);
        printf("\nOutput matrix: \n");
        print(S);
        draw(S);
    }
    break;
    case 3:
    {
        float rotate[3][3], theta;
        printf("\nEnter the angle in degree: ");
        scanf("%f", &theta);
        theta = theta * (3.14 / 180);
        rotate[0][0] = cos(theta);
        rotate[1][1] = cos(theta);
        rotate[0][2] = 0;
        rotate[1][2] = 0;
        rotate[2][0] = 0;
        rotate[2][1] = 0;
        rotate[2][2] = 1;
        rotate[0][1] = sin(theta);
        rotate[1][0] = -sin(theta);
        printf("\nRotation matrix: \n");
        print(rotate);
```

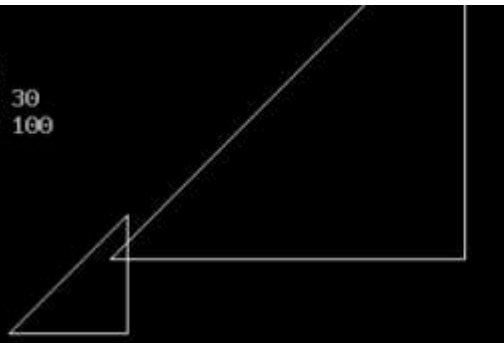
```
        multiply(obj, rotate, r);
        printf("\nOuput matrix: \n");
        print(r);
        draw(r);
    }
    break;
    case 4:
    {
        float ref[3][3];
        for (i = 0; i < 3; i++)
        {
            for (j = 0; j < 3; j++)
            {
                if (i == j)
                    ref[i][j] = 1;
                else
                    ref[i][j] = 0;
            }
        }
        ref[0][0] = ref[1][1] = -1;
        printf("\nReflection matrix: \n");
        print(ref);
        multiply(obj, ref, R);
        printf("\nOutput matrix: \n");
        print(R);
        draw(R);
    }
    break;
    case 5:
    {
        float shear[3][3], shx, shy;
        printf("\nEnter shx and shy: \n");
        scanf("%f%f", &shx, &shy);
        for (i = 0; i < 3; i++)
        {
            for (j = 0; j < 3; j++)
            {
                if (i == j)
                    shear[i][j] = 1;
                else
                    shear[i][j] = 0;
            }
        }
        shear[0][1] = shy;
        shear[1][0] = shx;
        printf("\nShearing matrix: \n");
        print(shear);
        multiply(obj, shear, Sh);
        printf("Output matrix: \n");
        print(Sh);
```

```
        draw(Sh);
    }
    break;
    default:
        printf("\nInvalid choice!");
    }
    getch();
}
```

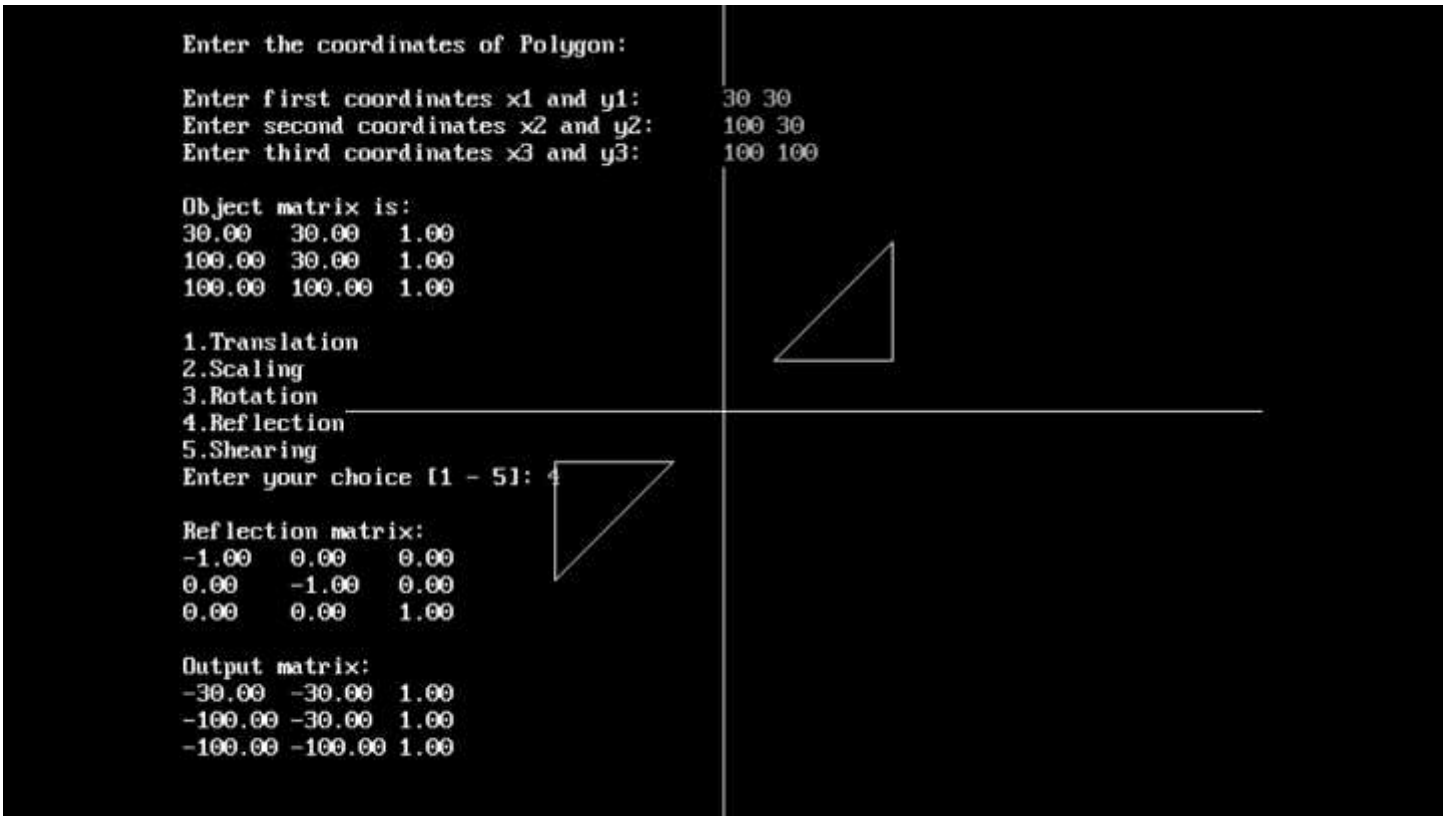
Output :



```
Enter the coordinates of Polygon:

Enter first coordinates x1 and y1:    30 30
Enter second coordinates x2 and y2:     100 30
Enter third coordinates x3 and y3:      100 100

Object matrix is:
30.00   30.00   1.00
100.00  30.00   1.00
100.00  100.00  1.00

1.Translation
2.Scaling
3.Rotation
4.Reflection
5.Shearing
Enter your choice [1 - 5]: 2

Enter values for sx and sy: 3 3

Scaling matrix:
3.00    0.00    0.00
0.00    3.00    0.00
0.00    0.00    1.00

Output matrix:
90.00   90.00   1.00
300.00  90.00   1.00
300.00  300.00  1.00
```



```
Enter the coordinates of Polygon:

Enter first coordinates x1 and y1:      30 30
Enter second coordinates x2 and y2:     100 30
Enter third coordinates x3 and y3:      100 100

Object matrix is:
30.00   30.00   1.00
100.00  30.00   1.00
100.00  100.00  1.00

1.Translation
2.Scaling
3.Rotation
4.Reflection
5.Shearing
Enter your choice [1 - 5]: 3

Enter the angle in degree: 90

Rotation matrix:
0.00    1.00    0.00
-1.00   0.00    0.00
0.00    0.00    1.00

Ouput matrix:
-29.98  30.02   1.00
-29.92  100.02  1.00
-99.92  100.08  1.00
```


Enter the coordinates of Polygon:
Enter first coordinates x1 and y1: 30 30
Enter second coordinates x2 and y2: 100 30
Enter third coordinates x3 and y3: 100 100
Object matrix is:
30.00 30.00 1.00
100.00 30.00 1.00
100.00 100.00 1.00
1.Translation
2.Scaling
3.Rotation
4.Reflection
5.Shearing
Enter your choice [1 - 5]: 4
Reflection matrix:
-1.00 0.00 0.00
0.00 -1.00 0.00
0.00 0.00 1.00
Output matrix:
-30.00 -30.00 1.00
-100.00 -30.00 1.00
-100.00 -100.00 1.00

Experiment - 7

PAGE No:
DATE / /

Aim: Implement Bezier curve

Algorithm:

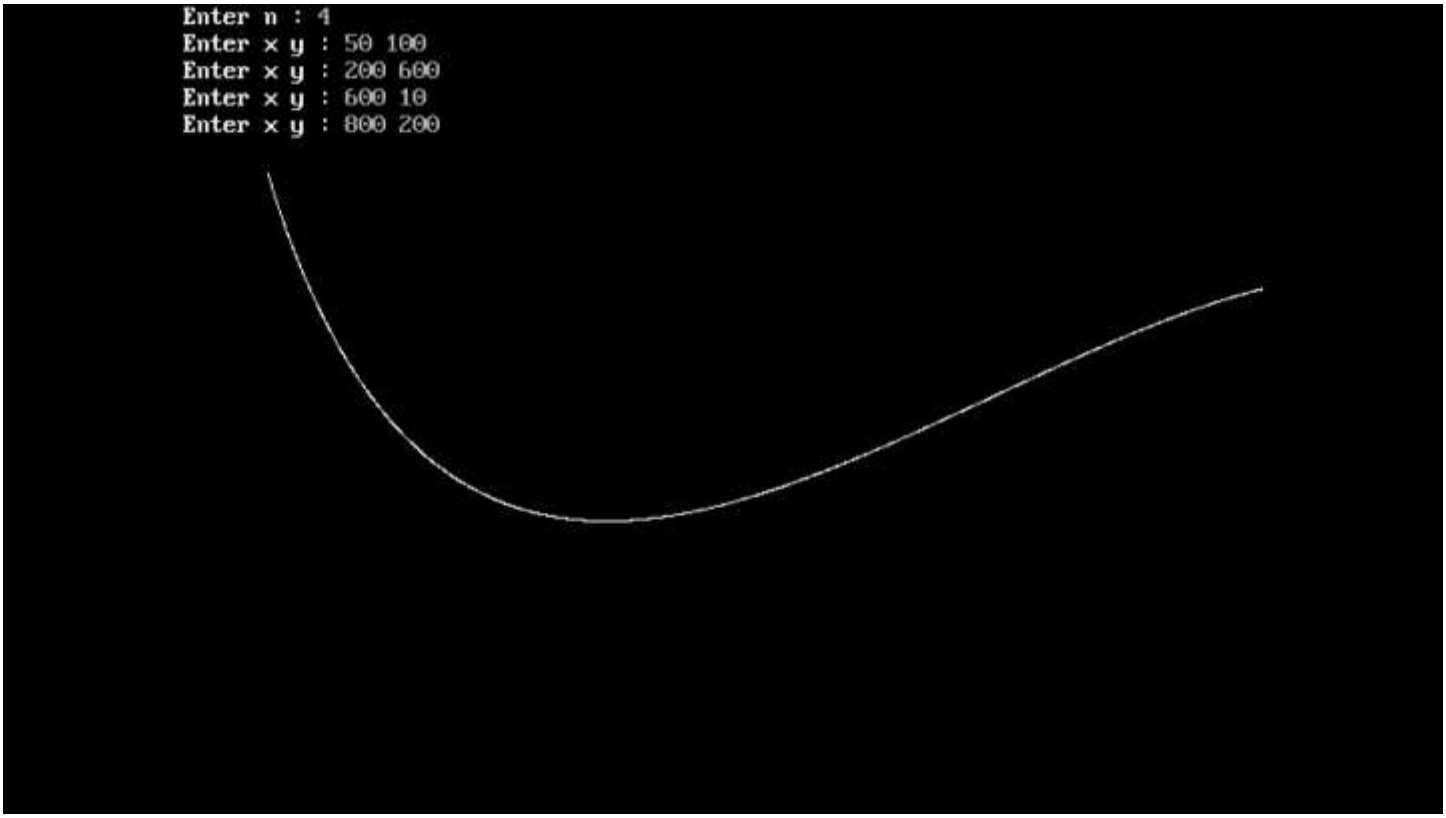
1. Get 4 control points say $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ & (x_4, y_4)
2. Divide the curve representation by point 1, 2, 3 & 4 in 2 sections.
3. $x_{12} = (x_1 + x_2)/2$
4. $y_{12} = (y_1 + y_2)/2$
5. $x_{23} = (x_2 + x_3)/2$
6. $y_{23} = (y_2 + y_3)/2$
7. $x_{34} = (x_3 + x_4)/2$
8. $y_{34} = (y_3 + y_4)/2$
9. $x_{123} = (x_{12} + x_{23})/2$
10. $y_{123} = (y_{12} + y_{23})/2$
11. $x_{1234} = (x_{123} + x_{234})/2$
12. $y_{1234} = (y_{123} + y_{1234})/2$

15. Repeat step 2 for 1, 12, 123, 1234.
16. Repeat step 3 until we have section very short.
17. That they can be replaced by lines.
18. Replace small section by lines.
19. Stop.

Program :

```
#include <stdio.h>
#include <graphics.h>
#include <math.h>
void main()
{
    int gd = DETECT, gm, i;
    double A, B, u;
    int x[10], y[10],n;
    printf("Enter n : ");
    scanf("%d",&n);
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    for (i = 0; i < n; i++)
    {
        printf("\nEnter x y: ");
        scanf("%d%d", &x[i], &y[i]);
    }
    for (u = 0.0; u <= 1.0; u += 0.0005)
    {
        A = x[0] * pow((1 - u), 3) + x[1] * 3 * u * pow((1 - u), 2) + x[2] * 3 * pow(u, 2) *
(1 - u) + x[3] * pow(u, 3);
        B = y[0] * pow((1 - u), 3) + y[1] * 3 * u * pow((1 - u), 2) + y[2] * 3 * pow(u, 2) *
(1 - u) + y[3] * pow(u, 3);
        putpixel(A, B, WHITE);
    }
    getch();
}
```

Output :


Enter n : 4
Enter x y : 50 100
Enter x y : 200 600
Enter x y : 600 10
Enter x y : 800 200

PAGE No.
DATE / /

Experiment-8

Aim: Implement koch curve for fractal generator.

Algorithm:

1. Get the number of iterations from user or N.
2. Compute total number of segments = $4N^2$
3. Save the traversing data in the form of matrix.
4. For (N iterations)
   {
5. compute the length of each segment
   = (1/3 * (current_iterations - 1)
6. compute current_segment_iteration =
   = ($4^{\wedge}$ current_iteration - 1))
8. For (range in current segment)

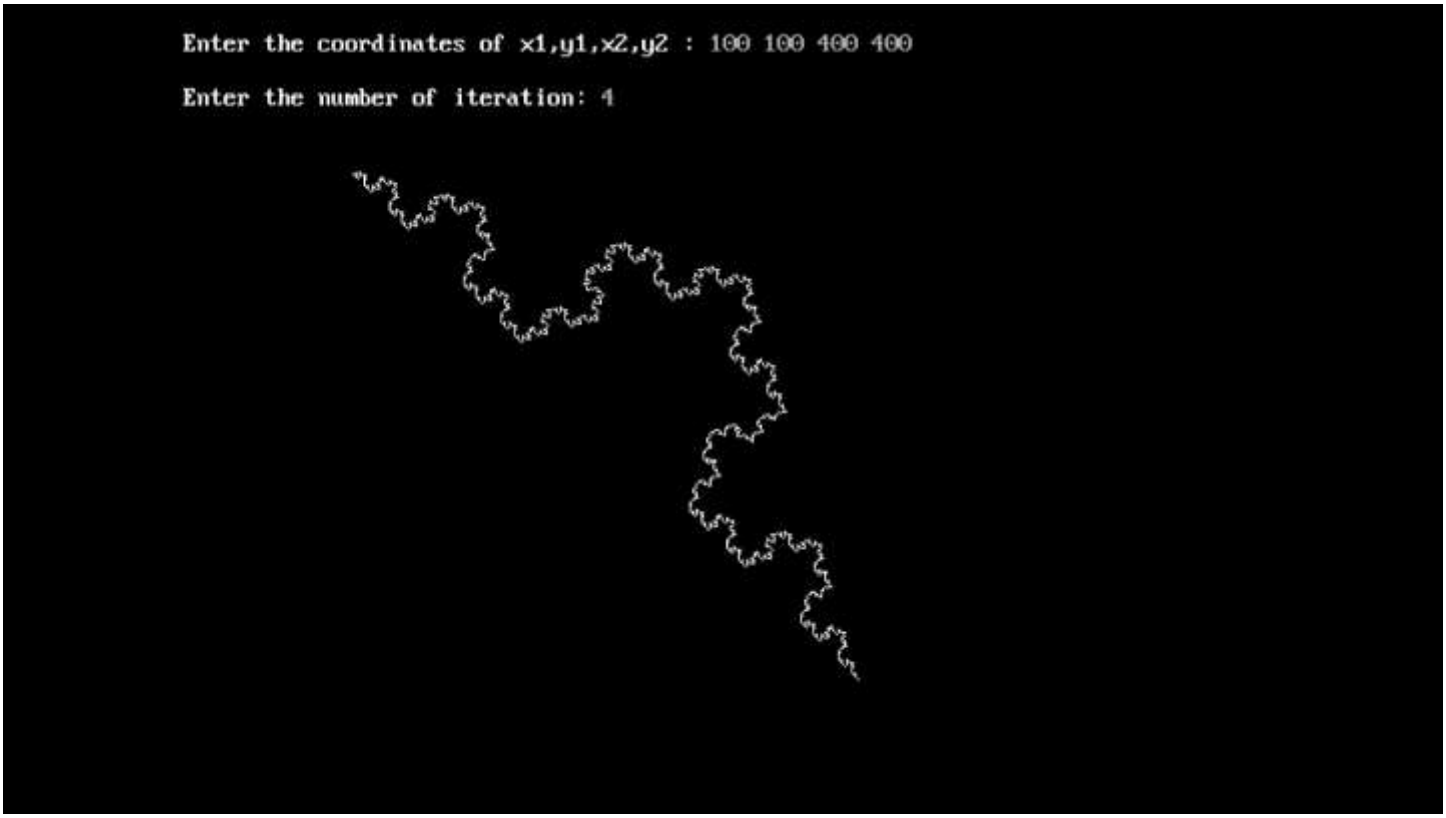
   perform matrix manipulations based on length of the geometry.

9. Store the value for future reference.
10. if not the last segment,
    delete the last segment.
11. update the value of x & y and store them seperately for transformation
12. plot the data generated.
13. Store generated data for implementation of offset

Program :

```
#include <graphics.h>
#include <conio.h>
#include <math.h>
void koch(int x1, int y1, int x2, int y2, int it)
{
    float angle = 60 * M_PI / 180;
    int x3 = (2 * x1 + x2) / 3;
    int y3 = (2 * y1 + y2) / 3;
    int x4 = (x1 + 2 * x2) / 3;
    int y4 = (y1 + 2 * y2) / 3;
    int x = x3 + (x4 - x3) * cos(angle) + (y4 - y3) * sin(angle);
    int y = y3 - (x4 - x3) * sin(angle) + (y4 - y3) * cos(angle);
    if (it > 0)
    {
        koch(x1, y1, x3, y3, it - 1);
        koch(x3, y3, x, y, it - 1);
        koch(x, y, x4, y4, it - 1);
        koch(x4, y4, x2, y2, it - 1);
    }
    else
    {
        line(x1, y1, x3, y3);
        line(x3, y3, x, y);
        line(x, y, x4, y4);
        line(x4, y4, x2, y2);
    }
}
int main(void)
{
    int gd = DETECT, gm;
    int x1, y1, x2, y2, n;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    printf("\nEnter the coordinates of x1,y1,x2,y2 : ");
    scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
    printf("\nEnter the number of iteration: ");
    scanf("%d", &n);
    koch(x1, y1, x2, y2, n);
    getch();
    return 0;
}
```

Output :


Enter the coordinates of x1,y1,x2,y2 : 100 100 400 400
Enter the number of iteration: 4

2003145

PAGE NO. 1
DATE

Experiment - 9

Aim: Implement Liang Barsky Line clipping algorithm in C.

Theory:

The Liang Barsky algorithm uses the parametric eqn of line and inequalities describing the range of window to determine intersection of line with clipping window. With these intersection it knows which portion of line to be drawn.

This algorithm is significantly efficient than cohen Sutherland algorithm. The idea behind this algorithm is to do as much testing as possible before drawing the intersections.

Algorithm:

1. Read 2 endpoints as (x_1, y_1) & (x_2, y_2)
2. Read two corners (left top & right bottom) of the clipping window as $(x_{wmin}, y_{wmin}, x_{wmax}, y_{wmax})$
3. Calculate values of parameters p_i & q_i for $i = 1$ to 4 such that

$p_1 = -dx$, $q_1 = x_1 - x_{wmin}$
$p_2 = dx$, $q_2 = x_{wmax} - x_1$
$p_3 = -dy$, $q_3 = y_1 - y_{wmin}$
$p_4 = dy$, $q_4 = y_{wmax} - y_1$.

4. If $p_i = 0$, then the is parallel to its boundary

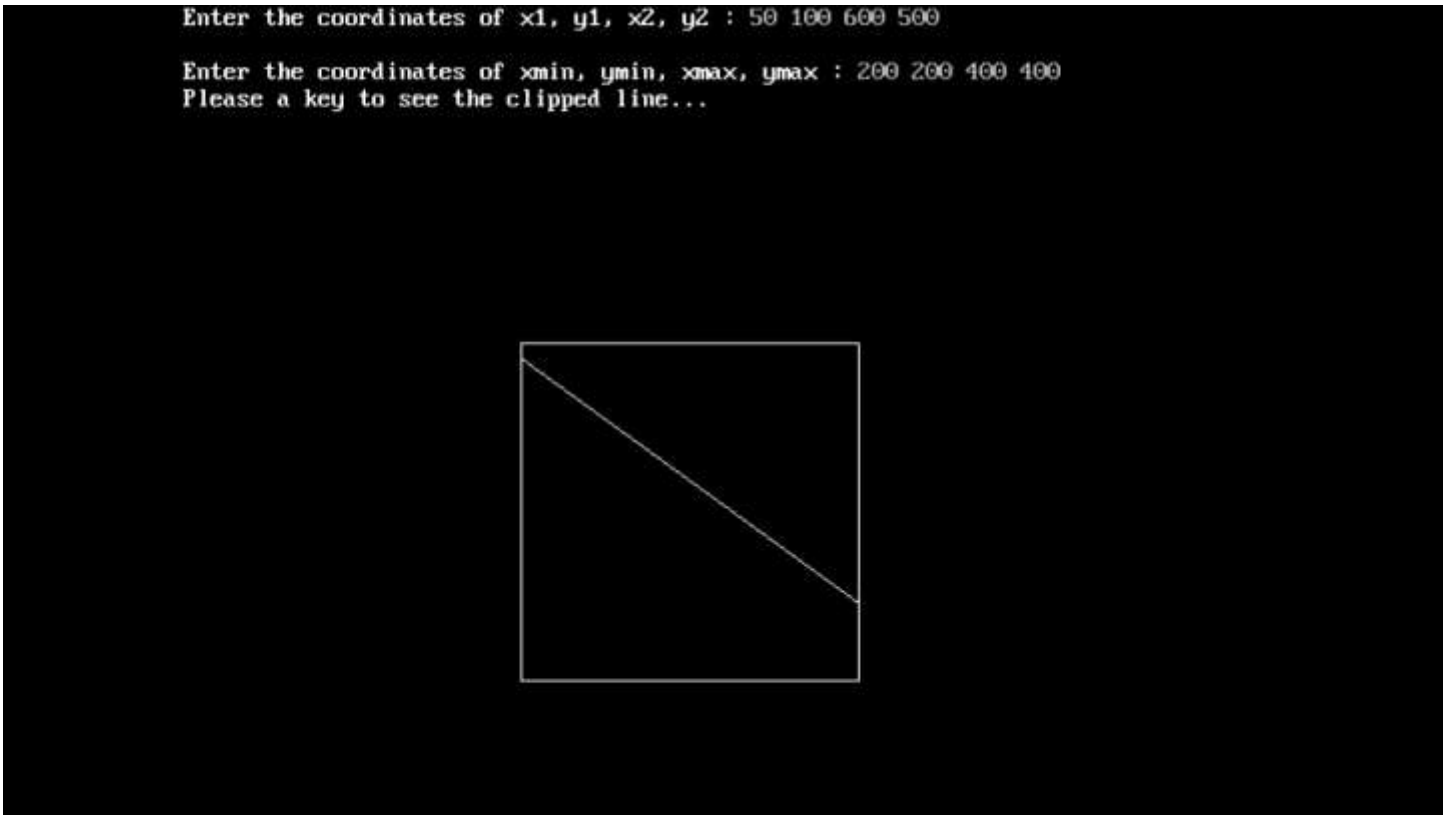
Program :

```
#include <stdio.h>
#include <graphics.h>
#include <math.h>
#include <dos.h>
void window(int, int, int, int);
void linedraw(int, int, int, int, int, int, int, int);

void main()
{
    int gd = DETECT, gm;
    int x1, y1, x2, y2, xmin, xmax, ymin, ymax;
    initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
    cleardevice();
    printf("Enter the coordinates of x1, y1, x2, y2 : ");
    scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
    printf("\nEnter the coordinates of xmin, ymin, xmax, ymax : ");
    scanf("%d%d%d%d", &xmin, &ymin, &xmax, &ymax);
    window(xmin, ymin, xmax, ymax);
    linedraw(x1, x2, y1, y2, xmin, xmax, ymin, ymax);
    getch();
    closegraph();
}

void window(int xmin, int ymin, int xmax, int ymax)
{
    rectangle(xmin, ymin, xmax, ymax);
    printf("Please a key to see the clipped line...");
    getch();
}

void linedraw(int x1, int x2, int y1, int y2, int xmin, int xmax, int ymin, int ymax)
{
    int a, b, i;
    float p[4], q[4], t1, t2, temp, xx1, xx2, yy1, yy2;
    a = x2 - x1;
    b = y2 - y1;
    p[0] = -a;
    p[1] = a;
    p[2] = -b;
    p[3] = b;
    q[0] = x1 - xmin;
    q[1] = xmax - x1;
    q[2] = y1 - ymin;
    q[3] = ymax - y1;
    for (i = 0; i < 4; i++)
```

```
{
    if (p[i] == 0)
    {
        printf("line is parallel");
        if (q[i] >= 0)
        {
            if (i < 2)
            {
                if (y1 < ymin)
                {
                    y1 = ymin;
                }
                if (y2 > ymax)
                {
                    y2 = ymax;
                }
                line(x1, y1, x2, y2);
            }
            if (i > 1)
            {
                if (x1 < xmin)
                {
                    x1 = xmin;
                }
                if (x2 > xmax)
                {
                    x2 = xmax;
                }
                line(x1, y1, x2, y2);
            }
        }
    }
}

t1 = 0.0;
t2 = 1.0;
for (i = 0; i < 4; i++)
{
    temp = q[i] / p[i];
    if (p[i] < 0)
    {
        if (t1 <= temp)
            t1 = temp;
    }
    else
    {
        if (t2 > temp)
            t2 = temp;
    }
}
```

```
    if (t1 < t2)
    {
        xx1 = x1 + t1 * p[1];
        xx2 = x1 + t2 * p[1];
        yy1 = y1 + t1 * p[3];
        yy2 = y1 + t2 * p[3];
        line(xx1, yy1, xx2, yy2);
    }
}
```

Output :



2003145

PAGE NO.
DATE / /

Experiment. 10

Aim: Implement Sutherland Hodgeman polygon clipping method in C.

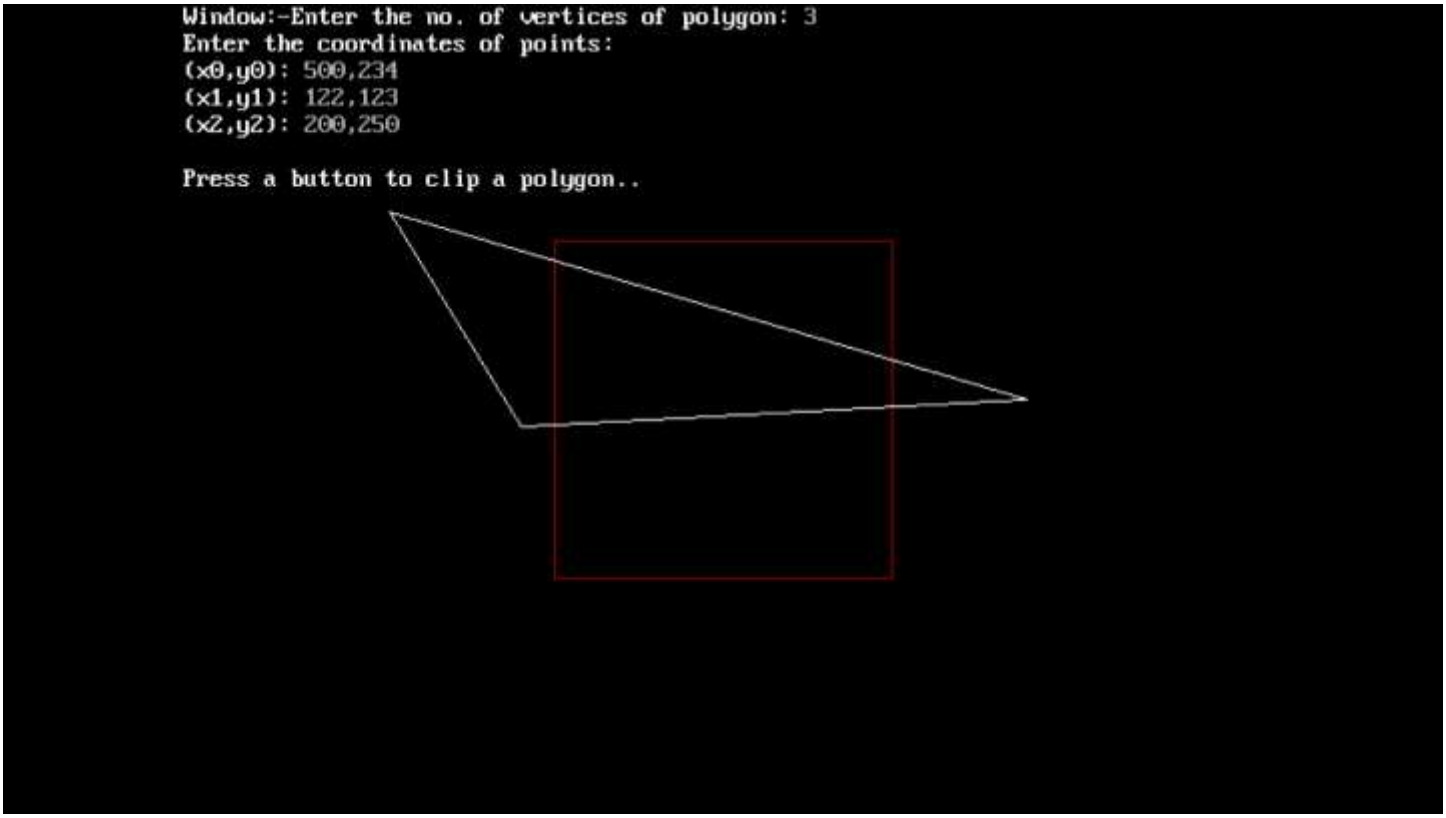
Algorithm:

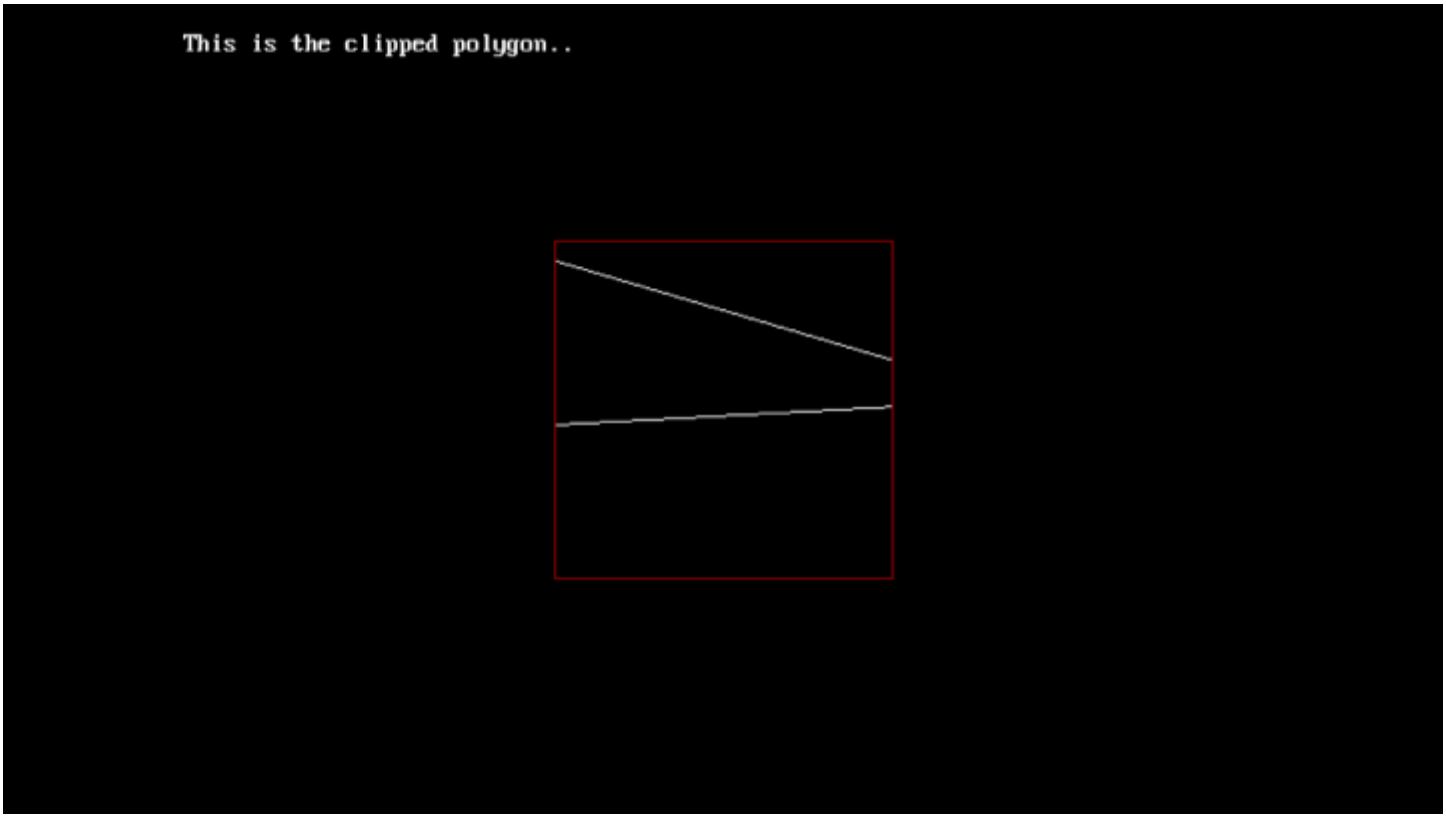
1. Input coordinates of all vertices of polygon.
2. Input coordinates of clipping window.
3. Consider the left edge of the window
4. Compare the vertice of each ~~window~~ polygon individually with the clipping pane.
5. Save the resulting intersections and vertices in new list of vertices according to four possible relationships between the edge and the clipping boundary.
6. Repeat step 4 & 5. for remaining edges of clipping window.
7. Each time the resultant list of vertices is successfully passed to process the next edge of the clippy window.
8. Stop.

Program :

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<stdlib.h>
int main()
{
    int gd,gm,n,*x,i,k=0;

    int w[]={220,140,420,140,420,340,220,340,220,140};
    detectgraph(&gd,&gm);
    initgraph(&gd,&gm,"c:\\turboc3\\bgi");
    printf("Window:-");
    setcolor(RED);
    drawpoly(5,w);
    printf("Enter the no. of vertices of polygon: ");
    scanf("%d",&n);
    x = malloc(n*2+1);
    printf("Enter the coordinates of points:\n");
    k=0;
    for(i=0;i<n*2;i+=2)
    {
        printf("(x%d,y%d): ",k,k);
        scanf("%d,%d",&x[i],&x[i+1]);
        k++;
    }
    x[n*2]=x[0];
    x[n*2+1]=x[1];
    setcolor(WHITE);
    drawpoly(n+1,x);
    printf("\nPress a button to clip a polygon..");
    getch();
    setcolor(RED);
    drawpoly(5,w);
    setfillstyle(SOLID_FILL,BLACK);
    floodfill(2,2,RED);
    gotoxy(1,1);
    printf("\nThis is the clipped polygon..");
    getch();

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

Output :


Window:-Enter the no. of vertices of polygon: 3
Enter the coordinates of points:
(x0,y0): 500,234
(x1,y1): 122,123
(x2,y2): 200,250
Press a button to clip a polygon..


This is the clipped polygon..

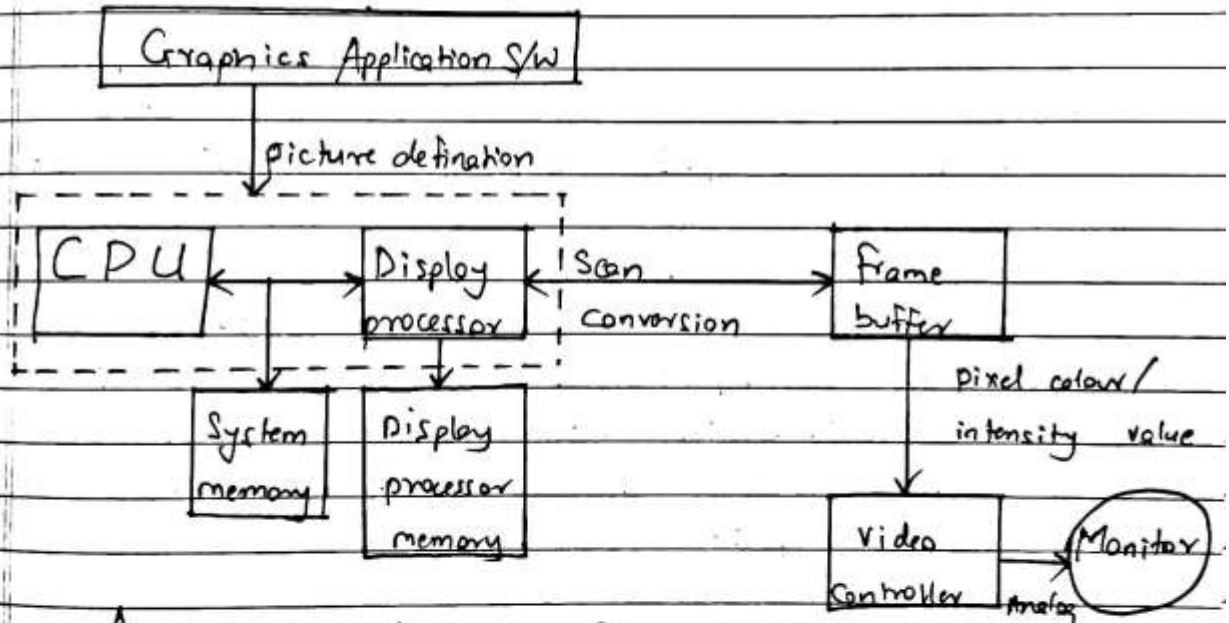
Idris Rattemwala
2003145

DATE

C.G. ASSIGNMENT. 1

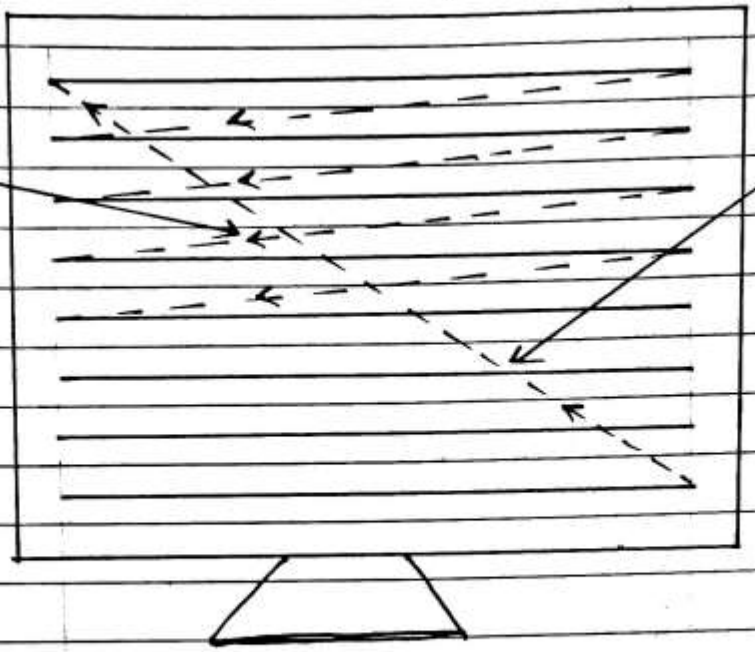
i) Architecture of Raster Scan Display:

Raster scan display basically employs a cathode ray tube (CRT) or an LCD panel for display. The Raster Scan Display viewing surface is coated with a layer of arrayed phosphor dots. At the back of the CRT is a set of electric guns (cathodes) that produce a controlled stream of electron say electron beam. The phosphor material emits light when struck with these high energy electrons. The architecture of Raster scan display is given below:



Architecture of Raster Scan Display.

The frequency of and intensity of the emitting light depends on the type of phosphor material used and the energy of electrons. To produce a picture on a screen, these directed electron beams start at the top of the screen. It scans rapidly from left to right along one row of phosphor dots.



ii)

Some of the applications of computer graphics are:

1) Computer aided drawing:

Designing of buildings, automobiles, aircraft is done with the help of computer aided drawings, this helps in providing minute details to the drawing and producing more accurate and sharp drawings with better specifications.

2) Computer art:

Using computer graphics we can create fine and commercial art which includes animation packages, paint packages. These packages provide facility for designing object shapes and specifing object motion. Cartoon drawing painting, logo design can also be done.

3) Presentation graphics :

For the preparation of reports or summarizing the financial, statistical, mathematical, scientific, economic data for research reports, managerial reports, moreover creation of bar graphs, pie charts, pie charts, can be done using the tools present in the computer graphics.

4) Entertainment :

Computer graphics finds its major part of its utility in the movie industry and gaming industries, used for creating motion pictures, music, videos, Television shows, cartoon animations films. In the game industry where the focus and the interactivity are the key players computer graphics helps in providing such features in efficient way.

5) Education :

Computer generated models are extremely useful for teaching ~~numerous~~ huge number of concepts and fundamentals in easy to understand and learn manner. Using computer graphics many educational models can be created through which more interest can be generated among the students who are learning the subject.

6) Image processing :

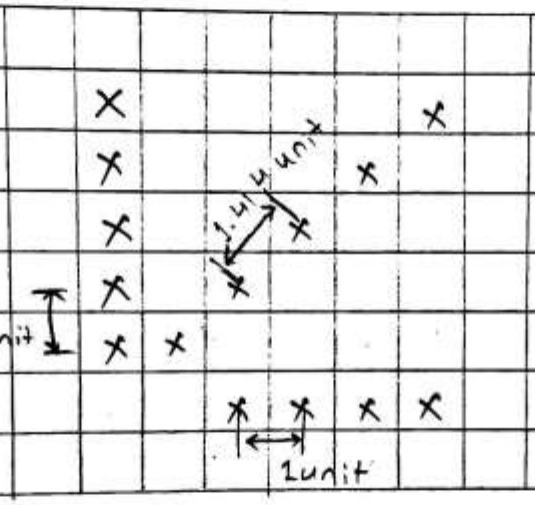
Various kinds of photographs or images require editing inorder to be used in different places, processing of the existing images in to refined ones for better interpretation is one of the many applications of the computer graphics.

iii)

Raster Scan	Random Scan
1. Raster scan produces jagged lines that are plotted as discrete point sets	1. Random scan produces smooth lines, because CRT beam follows the line path directly.
2. It is less expensive	2. It is more expensive.
3. Modification is difficult	3. Modification is easy.
4. It's resolution is low because picture definition is stored as a set of intensity value for all screen points.	4. Its resolution is high because pictures definition is stored as a set of line drawing instructions.
5. Solid pattern is easy to fill	5. Solid pattern is difficult to fill.
6. It is suitable for realistic display and well suited for displaying shading and colour areas.	6. It is suitable for engineering and sci. drawings, restricted to line drawing applications
7. Shadow mask technology is used	7. Beam penetration technology is used.
8. Image is displayed by scanning the whole one.	8. Image is displayed by the beam along the vectors
9. The refresh rate is independent of picture complexity.	9. The refresh rate depends directly on picture complexity

iv) Aliasing:

No matter how good a line or circle drawing algorithm is it is impossible to avoid giving most discrete line & circle a staircase (or jaggy) effect. They will not look straight. Jagged edges are caused by limitation in computer screen whether it's a CRT or LCD/TFT screen. Monitors are capable of producing nearly perfect straight lines either horizontally or vertically, but when it comes to an inclined line, it is not able to produce without jagged edges.



Relative distance between two pixels in possible cases.

• Types of antidliasing

There are mainly 2 types of antialiasing techinques:

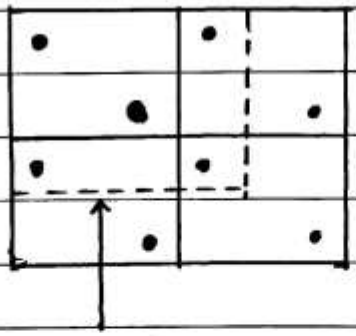
1) Super sampling
2) Multi sampling.

1) Super sampling: Its a method of anti aliasing by taking the colour of corner pixels and creating what would be the average colour. which is then displayed on the screen's pixel. By doing this you are smudging the image, and averaging out the colour of the corner.

Effectively the super sampling renders the scence 4 times larger and then scales it down, once calculations are complete. This system has massive performance hit and gives the best results.

2) Multisampling:

Multisampling is more efficient but slightly less pricy than antialiasing. It takes multiple samples for each pixel. In for example, the quincunx system ~~has~~ takes 4 samples in the corners and 1 sample in the middle. Each of these samples are given weight The corners are given 1/8 each & centre a weight of 1/2 - The colour of pixel is then determined by super sampling



An Example of multisampling

Idris Rotkewale
C31-2003145
Idris

C.G. Assingment 2

DATE / /

i) Traditional animation techiques :

1) Life is given to any object using animation. Animations are of 2 types ie i) Computer assisted
ii) Computer generated.
and can be presented either by video or film.

2) Traditional animation techiques are the following are animation techiques.
Traditional animation, Key framing, procedural, behavior, performance based, physically based

3) Traditional Animation refers to animation hand written on a piece of paper. It was the method used for most of production over 20th century.

4) An animator draws the character, layout and background on a paper, drawing each frame slightly different.

5) Once all the animations are drawn on paper it is then photocopied or retraced onto transparent acetate sheets cels.

2) Differentiate between parallel and perspective projection

Parallel Projection	Perspective Projection
1. It represents any given object in a different way as we view it on a telescope.	1. It represents any given object in 3 dimensional manner.
2. The projector is parallel.	2. The projector is not parallel.
3. It does not alter the shape or size of object.	3. The object stays far away appears to be smaller, while one's near to eyes appears big.
4. Distance of the given object is infinite from centre of projection.	4. The distance of given object is finite from centre of projection
5. It can provide user accurate view of given object.	5. The shape & size tend to differ from its origination.
6. The parallel lines of projections are parallel to each other.	6. The perspective lines are not parallel to each other.
7. It has 2 types: • Oblique • Orthographic	7. It has 3 types: • One point • two point • three point.

8) Explain B-spline curve.

→ i) A B-spline curve is defined as linear combination of control points & B-spline basis functions.

ii) The control points are called de Boor point. The basis funct is defined on a knot vector where there are n+k+1 elements, i.e. no. of control points n+1 plus order of the curve.

iii) Each knot span $t_i \leq t \leq t_{i+1}$ is mapped onto polynomial curve between 2 succ. pts & $r(t_{i+1})$

iv) Normalization of knot vector, covers the interval [0,1]. It is helpful in improving numerical accuracy in floating point arithmetic computation.

Properties of B-spline curve -

i) Geometry invariance property:
Partition of unity property of B-spline assures invariance of shape of B-spline curve under translation & rotation.

ii) Convex hull property:
It applies locally so that a span lies within the convex hull of control points that affect it.

iii) Local support property:
A single span of B-spline is controlled only by control pts & any control pts

affect spans.

iv) B-spline to Bezier property:
A Bezier curve of order (degree k-1) is B-spline curve with no internal knots & end knots repeated k times.

4) Explain window to viewport transformation.

→ i) Window to viewport transformation is process of transforming 2D world-coordinate objects to device co-ordinates.

ii) Objects inside are mapped to viewport which is the area on screen where world coordinates are mapped to be displayed.

iii) World coordinate is the cartesian co-ordinate wrt which are define the diagram like X_{wmin}, X_{wmax}, Y_{wmin}, Y_{wmax}.

iv) Device coordinate is screen coordinate where objects are to be displayed like X_{Vmin}, X_{Vmax}, Y_{Vmin}, Y_{Vmax}.

v) Window is the area on world coordinate selected for display.

vi) Viewport is area on device coordinate where graphics is to be displayed.

Math. calculation of Window to viewport -

1) It may be possible that size of viewport is much smaller or greater than window

In such cases, we need to increase or decrease size of window & for this we need mathematical calculations.

Normalized pt on window $\left(\frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}}, \frac{y_w - y_{wmin}}{y_{wmax} - y_{wmin}}\right)$

Normalized pt on vport $\left(\frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}}, \frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}}\right)$

Relative position of obj in window = viewport

for x coordinate -

$$\frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}} = \frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}}$$

for y coordinate -

$$\frac{y_w - y_{wmin}}{y_{wmax} - y_{wmin}} = \frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}}$$

After calculating for x & y coordinate we get,

$$x_v = x_{vmin} + (x_w - x_{wmin}) s_x$$
$$y_v = y_{vmin} + (y_w - y_{wmin}) s_y$$

s_x = scaling factor of x, s_y = scaling factor of y

$$s_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}$$

$$s_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}$$

PACMAN GAME

Idris Ratlamwala - 2003145

Lavin Rupani - 2003147

Priyansh Salian - 2003148

Description :

The goal of the game is to collect all the points in the maze and avoid the ghosts. The Pacman is animated in two ways: his position in the maze and his body. We animate his body with four images, depending on the direction. The animation is used to create the illusion of Pacman opening and closing his mouth. The maze consists of 15x15 squares. The structure of the maze is based on a simple array of integers. Pacman has three lives. We also count the score.

The Pacman is controlled with the cursor keys. The Esc key finishes the game, the Pause key pauses it.

An array stores the level data which provide information out of which we create the corners and the points. Number 1 is a left corner. Numbers 2, 4 and 8 represent top, right, and bottom corners respectively. Number 16 is a point. These numbers can be added, for example number 19 in the upper left corner means that the square will have top and left borders and a point (16 + 2 + 1).

There are four possible directions for a Pacman. There are four images for all directions. The images are used to animate Pacman opening and closing his mouth.

The drawMaze() method draws the maze out of the numbers in the screenData array. Number 1 is a left border, 2 is a top border, 4 is a right border, 8 is a bottom border and 16 is a point. We simply go through all 225 squares in the maze. For example we have 9 in the screenData array. We have the first bit (1) and the fourth bit (8) set. So we draw a bottom and a left border on this particular square.

Program :

Main class (Pacman.java) :

```
package pacman;

import javax.swing.JFrame;

public class Pacman extends JFrame{

    private static final long serialVersionUID = 1L;


    public Pacman() {
        add(new Model());
    }

    public static void main(String[] args) {
        Pacman pac = new Pacman();
        pac.setVisible(true);
        pac.setTitle("Pacman");
        pac.setSize(380,420);
        pac.setDefaultCloseOperation(EXIT_ON_CLOSE);
        pac.setLocationRelativeTo(null);

    }

}
```

Model.java class :

```
package pacman;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import javax.swing.ImageIcon;
import javax.swing.JPanel;
```

```
import javax.swing.Timer;

public class Model extends JPanel implements ActionListener {

    private Dimension d;
    private final Font smallFont = new Font("Arial", Font.BOLD, 14);
    private boolean inGame = false;
    private boolean dying = false;

    private final int BLOCK_SIZE = 24;
    private final int N_BLOCKS = 15;
    private final int SCREEN_SIZE = N_BLOCKS * BLOCK_SIZE;
    private final int MAX_GHOSTS = 12;
    private final int PACMAN_SPEED = 6;

    private int N_GHOSTS = 6;
    private int lives, score;
    private int[] dx, dy;
    private int[] ghost_x, ghost_y, ghost_dx, ghost_dy, ghostSpeed;

    private Image heart, ghost;
    private Image up, down, left, right;

    private int pacman_x, pacman_y, pacmand_x, pacmand_y;
    private int req_dx, req_dy;

    private final short levelData[] = {
        19, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 22,
        17, 16, 16, 16, 16, 24, 16, 16, 16, 16, 16, 16, 16, 16, 20,
        25, 24, 24, 24, 28, 0, 17, 16, 16, 16, 16, 16, 16, 16, 20,
        0,  0,  0,  0,  0,  0, 17, 16, 16, 16, 16, 16, 16, 16, 20,
        19, 18, 18, 18, 18, 18, 16, 16, 16, 16, 24, 24, 24, 24, 20,
        17, 16, 16, 16, 16, 16, 16, 16, 16, 20, 0,  0,  0,   0, 21,
        17, 16, 16, 16, 16, 16, 16, 16, 16, 20, 0,  0,  0,   0, 21,
        17, 16, 16, 16, 24, 16, 16, 16, 16, 20, 0,  0,  0,   0, 21,
        17, 16, 16, 20, 0, 17, 16, 16, 16, 16, 18, 18, 18, 18, 20,
        17, 24, 24, 28, 0, 25, 24, 24, 16, 16, 16, 16, 16, 16, 20,
        21, 0,  0,  0,  0,  0,  0,   0, 17, 16, 16, 16, 16, 16, 20,
        17, 18, 18, 22, 0, 19, 18, 18, 16, 16, 16, 16, 16, 16, 20,
        17, 16, 16, 20, 0, 17, 16, 16, 16, 16, 16, 16, 16, 16, 20,
        17, 16, 16, 20, 0, 17, 16, 16, 16, 16, 16, 16, 16, 16, 20,
        25, 24, 24, 24, 26, 24, 24, 24, 24, 24, 24, 24, 24, 28
    };

    private final int validSpeeds[] = {1, 2, 3, 4, 6, 8};
    private final int maxSpeed = 6;

    private int currentSpeed = 3;
    private short[] screenData;
    private Timer timer;
```

```
        public Model() {

            loadImages();
            initVariables();
            addKeyListener(new TAdapter());
            setFocusable(true);
            initGame();
        }


        private void loadImages() {
            //down = new ImageIcon("/src/images/down.gif").getImage();
            down = new ImageIcon("C:\\Users\\IsmailRatlamwala\\Downloads\\Pacman-
master\\images\\down.gif").getImage();
            up = new ImageIcon("C:\\Users\\IsmailRatlamwala\\Downloads\\Pacman-
master\\images\\up.gif").getImage();
            left = new ImageIcon("C:\\Users\\IsmailRatlamwala\\Downloads\\Pacman-
master\\images\\left.gif").getImage();
            right = new ImageIcon("C:\\Users\\IsmailRatlamwala\\Downloads\\Pacman-
master\\images\\right.gif").getImage();
            ghost = new ImageIcon("C:\\Users\\IsmailRatlamwala\\Downloads\\Pacman-
master\\images\\ghost.gif").getImage();
            heart = new ImageIcon("C:\\Users\\IsmailRatlamwala\\Downloads\\Pacman-
master\\images\\heart.png").getImage();

        }
        private void initVariables() {

            screenData = new short[N_BLOCKS * N_BLOCKS];
            d = new Dimension(400, 400);
            ghost_x = new int[MAX_GHOSTS];
            ghost_dx = new int[MAX_GHOSTS];
            ghost_y = new int[MAX_GHOSTS];
            ghost_dy = new int[MAX_GHOSTS];
            ghostSpeed = new int[MAX_GHOSTS];
            dx = new int[4];
            dy = new int[4];

            timer = new Timer(40, this);
            timer.start();
        }

        private void playGame(Graphics2D g2d) {

            if (dying) {

                death();

            } else {
```

```
            movePacman();
            drawPacman(g2d);
            moveGhosts(g2d);
            checkMaze();
        }
    }

    private void showIntroScreen(Graphics2D g2d) {

        String start = "Press SPACE to start";
        g2d.setColor(Color.yellow);
        g2d.drawString(start, (SCREEN_SIZE)/4, 150);
    }

    private void drawScore(Graphics2D g) {
        g.setFont(smallFont);
        g.setColor(new Color(5, 181, 79));
        String s = "Score: " + score;
        g.drawString(s, SCREEN_SIZE / 2 + 96, SCREEN_SIZE + 16);

        for (int i = 0; i < lives; i++) {
            g.drawImage(heart, i * 28 + 8, SCREEN_SIZE + 1, this);
        }
    }

    private void checkMaze() {

        int i = 0;
        boolean finished = true;

        while (i < N_BLOCKS * N_BLOCKS && finished) {

            if ((screenData[i]) != 0) {
                finished = false;
            }

            i++;
        }

        if (finished) {

            score += 50;

            if (N_GHOSTS < MAX_GHOSTS) {
                N_GHOSTS++;
            }

            if (currentSpeed < maxSpeed) {
                currentSpeed++;
```

```
                }

                initLevel();
            }
        }

        private void death() {

            lives--;

            if (lives == 0) {
                inGame = false;
            }

            continueLevel();
        }

        private void moveGhosts(Graphics2D g2d) {

            int pos;
            int count;

            for (int i = 0; i < N_GHOSTS; i++) {
                if (ghost_x[i] % BLOCK_SIZE == 0 && ghost_y[i] % BLOCK_SIZE == 0) {
                    pos = ghost_x[i] / BLOCK_SIZE + N_BLOCKS * (int) (ghost_y[i] /
BLOCK_SIZE);

                    count = 0;

                    if ((screenData[pos] & 1) == 0 && ghost_dx[i] != 1) {
                        dx[count] = -1;
                        dy[count] = 0;
                        count++;
                    }

                    if ((screenData[pos] & 2) == 0 && ghost_dy[i] != 1) {
                        dx[count] = 0;
                        dy[count] = -1;
                        count++;
                    }

                    if ((screenData[pos] & 4) == 0 && ghost_dx[i] != -1) {
                        dx[count] = 1;
                        dy[count] = 0;
                        count++;
                    }

                    if ((screenData[pos] & 8) == 0 && ghost_dy[i] != -1) {
                        dx[count] = 0;
                        dy[count] = 1;
```

```
                    count++;
                }

                if (count == 0) {

                    if ((screenData[pos] & 15) == 15) {
                        ghost_dx[i] = 0;
                        ghost_dy[i] = 0;
                    } else {
                        ghost_dx[i] = -ghost_dx[i];
                        ghost_dy[i] = -ghost_dy[i];
                    }

                } else {

                    count = (int) (Math.random() * count);

                    if (count > 3) {
                        count = 3;
                    }

                    ghost_dx[i] = dx[count];
                    ghost_dy[i] = dy[count];
                }

            }

            ghost_x[i] = ghost_x[i] + (ghost_dx[i] * ghostSpeed[i]);
            ghost_y[i] = ghost_y[i] + (ghost_dy[i] * ghostSpeed[i]);
            drawGhost(g2d, ghost_x[i] + 1, ghost_y[i] + 1);

            if (pacman_x > (ghost_x[i] - 12) && pacman_x < (ghost_x[i] + 12)
                    && pacman_y > (ghost_y[i] - 12) && pacman_y < (ghost_y[i] + 12)
                    && inGame) {

                dying = true;
            }
        }
    }

    private void drawGhost(Graphics2D g2d, int x, int y) {
        g2d.drawImage(ghost, x, y, this);
        }

    private void movePacman() {

        int pos;
        short ch;

        if (pacman_x % BLOCK_SIZE == 0 && pacman_y % BLOCK_SIZE == 0) {
```

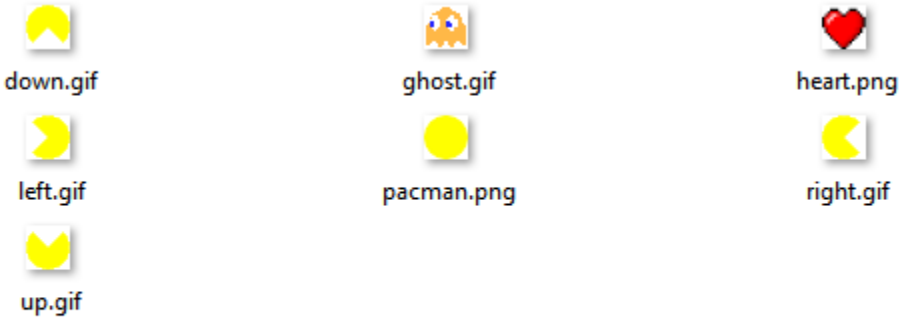
```
            pos = pacman_x / BLOCK_SIZE + N_BLOCKS * (int) (pacman_y / BLOCK_SIZE);
            ch = screenData[pos];

            if ((ch & 16) != 0) {
                screenData[pos] = (short) (ch & 15);
                score++;
            }

            if (req_dx != 0 || req_dy != 0) {
                if (!((req_dx == -1 && req_dy == 0 && (ch & 1) != 0)
                        || (req_dx == 1 && req_dy == 0 && (ch & 4) != 0)
                        || (req_dx == 0 && req_dy == -1 && (ch & 2) != 0)
                        || (req_dx == 0 && req_dy == 1 && (ch & 8) != 0))) {
                    pacmand_x = req_dx;
                    pacmand_y = req_dy;
                }
            }

            // Check for standstill
            if ((pacmand_x == -1 && pacmand_y == 0 && (ch & 1) != 0)
                    || (pacmand_x == 1 && pacmand_y == 0 && (ch & 4) != 0)
                    || (pacmand_x == 0 && pacmand_y == -1 && (ch & 2) != 0)
                    || (pacmand_x == 0 && pacmand_y == 1 && (ch & 8) != 0)) {
                pacmand_x = 0;
                pacmand_y = 0;
            }
        }
        pacman_x = pacman_x + PACMAN_SPEED * pacmand_x;
        pacman_y = pacman_y + PACMAN_SPEED * pacmand_y;
    }

    private void drawPacman(Graphics2D g2d) {

        if (req_dx == -1) {
            g2d.drawImage(left, pacman_x + 1, pacman_y + 1, this);
        } else if (req_dx == 1) {
            g2d.drawImage(right, pacman_x + 1, pacman_y + 1, this);
        } else if (req_dy == -1) {
            g2d.drawImage(up, pacman_x + 1, pacman_y + 1, this);
        } else {
            g2d.drawImage(down, pacman_x + 1, pacman_y + 1, this);
        }
    }

    private void drawMaze(Graphics2D g2d) {

        short i = 0;
        int x, y;

        for (y = 0; y < SCREEN_SIZE; y += BLOCK_SIZE) {
```

```
            for (x = 0; x < SCREEN_SIZE; x += BLOCK_SIZE) {

                g2d.setColor(new Color(0,72,251));
                g2d.setStroke(new BasicStroke(5));

                if ((levelData[i] == 0)) {
                    g2d.fillRect(x, y, BLOCK_SIZE, BLOCK_SIZE);
                }

                if ((screenData[i] & 1) != 0) {
                    g2d.drawLine(x, y, x, y + BLOCK_SIZE - 1);
                }

                if ((screenData[i] & 2) != 0) {
                    g2d.drawLine(x, y, x + BLOCK_SIZE - 1, y);
                }

                if ((screenData[i] & 4) != 0) {
                    g2d.drawLine(x + BLOCK_SIZE - 1, y, x + BLOCK_SIZE - 1,
                            y + BLOCK_SIZE - 1);
                }

                if ((screenData[i] & 8) != 0) {
                    g2d.drawLine(x, y + BLOCK_SIZE - 1, x + BLOCK_SIZE - 1,
                            y + BLOCK_SIZE - 1);
                }

                if ((screenData[i] & 16) != 0) {
                    g2d.setColor(new Color(255,255,255));
                    g2d.fillOval(x + 10, y + 10, 6, 6);
                }

                i++;
            }
        }
    }

    private void initGame() {

        lives = 3;
        score = 0;
        initLevel();
        N_GHOSTS = 6;
        currentSpeed = 3;
    }

    private void initLevel() {

        int i;
        for (i = 0; i < N_BLOCKS * N_BLOCKS; i++) {
```

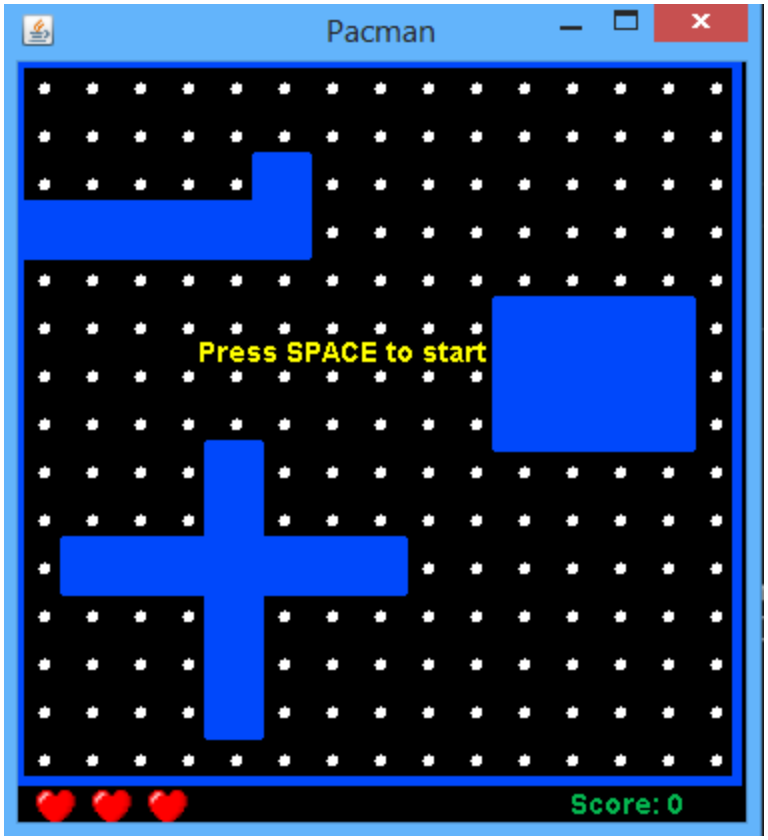
```
            screenData[i] = levelData[i];
        }

        continueLevel();
    }

    private void continueLevel() {

        int dx = 1;
        int random;

        for (int i = 0; i < N_GHOSTS; i++) {

            ghost_y[i] = 4 * BLOCK_SIZE; //start position
            ghost_x[i] = 4 * BLOCK_SIZE;
            ghost_dy[i] = 0;
            ghost_dx[i] = dx;
            dx = -dx;
            random = (int) (Math.random() * (currentSpeed + 1));

            if (random > currentSpeed) {
                random = currentSpeed;
            }

            ghostSpeed[i] = validSpeeds[random];
        }

        pacman_x = 7 * BLOCK_SIZE;  //start position
        pacman_y = 11 * BLOCK_SIZE;
        pacmand_x = 0;  //reset direction move
        pacmand_y = 0;
        req_dx = 0;     // reset direction controls
        req_dy = 0;
        dying = false;
    }


    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        Graphics2D g2d = (Graphics2D) g;

        g2d.setColor(Color.black);
        g2d.fillRect(0, 0, d.width, d.height);

        drawMaze(g2d);
        drawScore(g2d);

        if (inGame) {
            playGame(g2d);
```

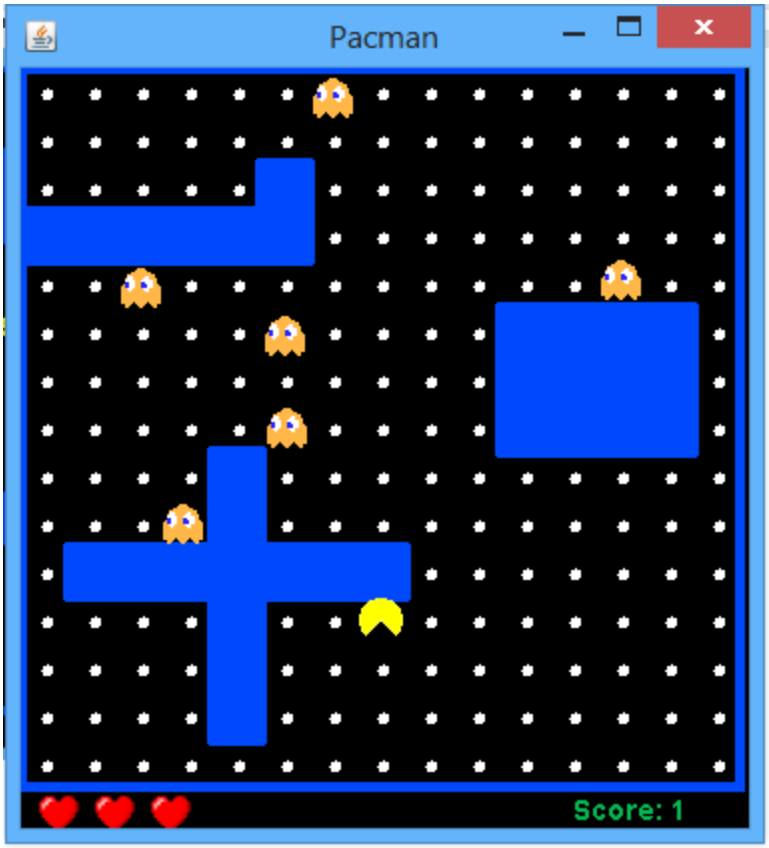
```
            } else {
                showIntroScreen(g2d);
            }

            Toolkit.getDefaultToolkit().sync();
            g2d.dispose();
        }


        //controls
        class TAdapter extends KeyAdapter {

            @Override
            public void keyPressed(KeyEvent e) {

                int key = e.getKeyCode();

                if (inGame) {
                    if (key == KeyEvent.VK_LEFT) {
                        req_dx = -1;
                        req_dy = 0;
                    } else if (key == KeyEvent.VK_RIGHT) {
                        req_dx = 1;
                        req_dy = 0;
                    } else if (key == KeyEvent.VK_UP) {
                        req_dx = 0;
                        req_dy = -1;
                    } else if (key == KeyEvent.VK_DOWN) {
                        req_dx = 0;
                        req_dy = 1;
                    } else if (key == KeyEvent.VK_ESCAPE && timer.isRunning()) {
                        inGame = false;
                    }
                } else {
                    if (key == KeyEvent.VK_SPACE) {
                        inGame = true;
                        initGame();
                    }
                }
            }
    }


        @Override
        public void actionPerformed(ActionEvent e) {
            repaint();
        }

        }
```

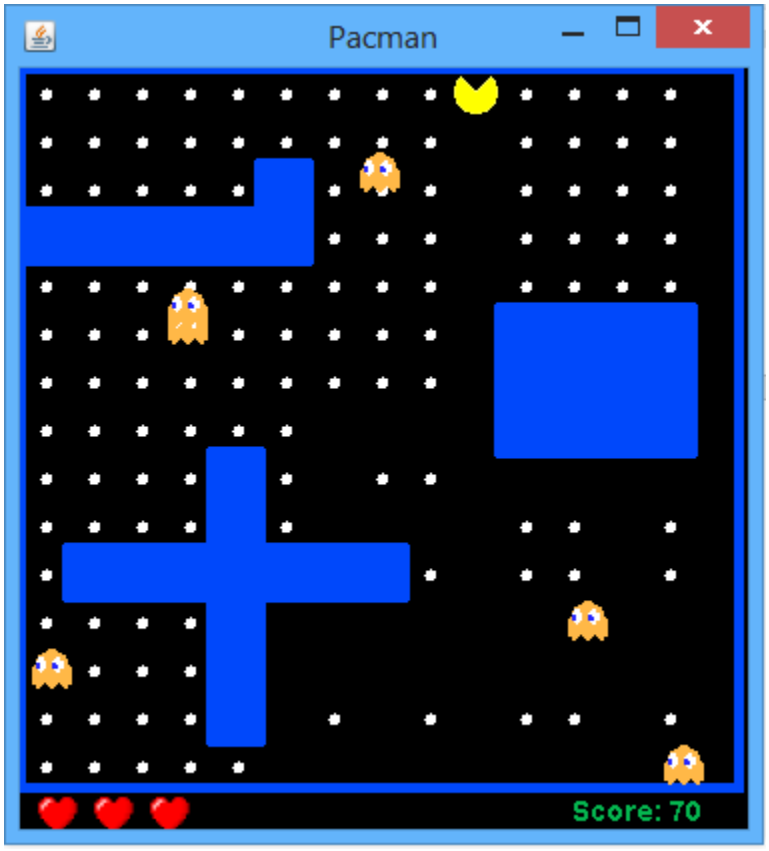
Images :

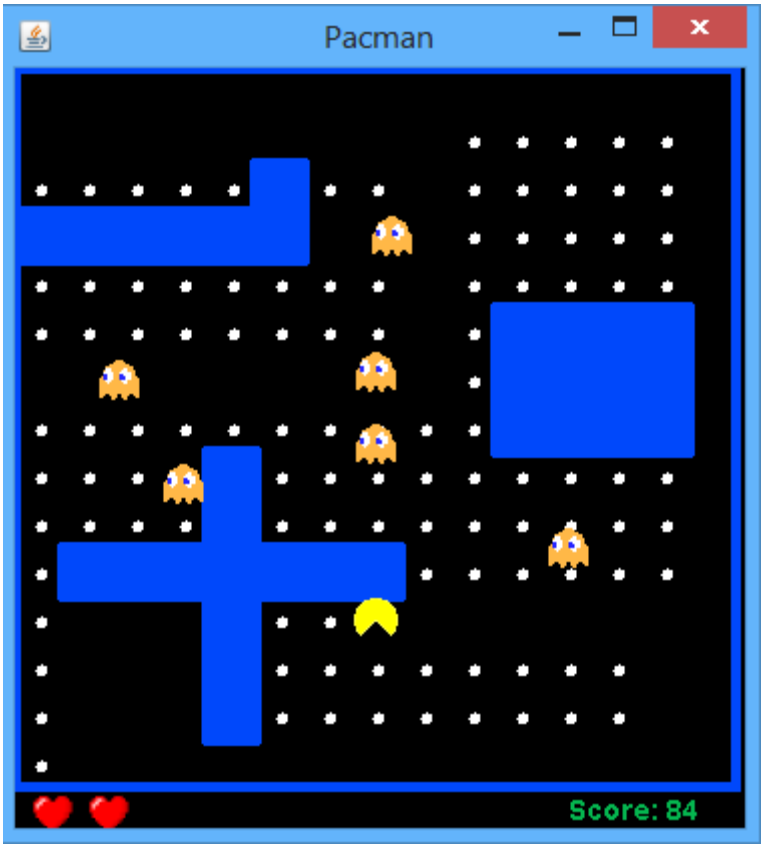


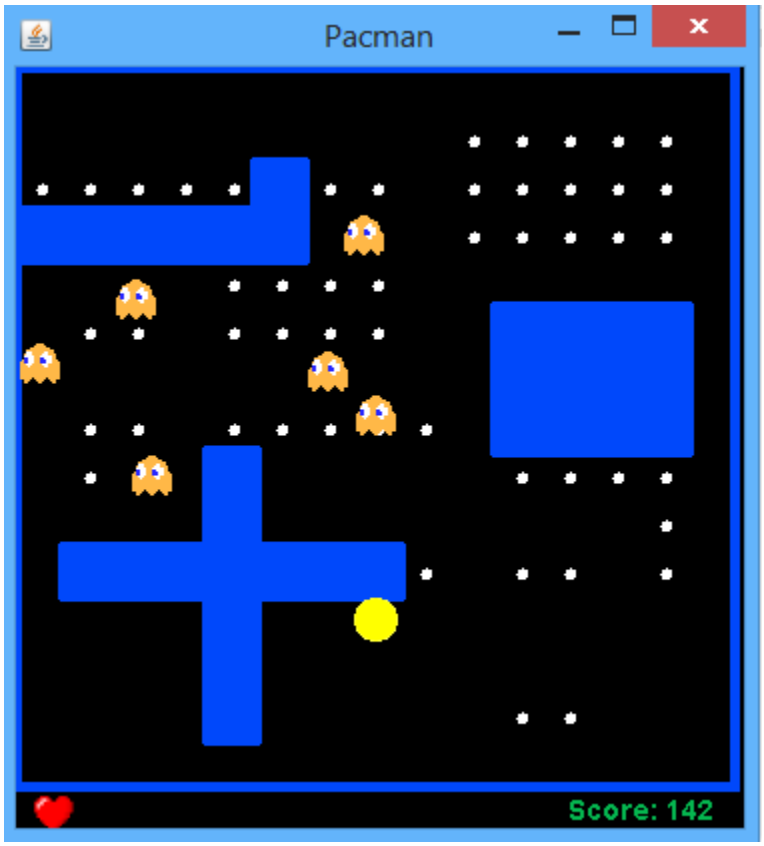
Output :

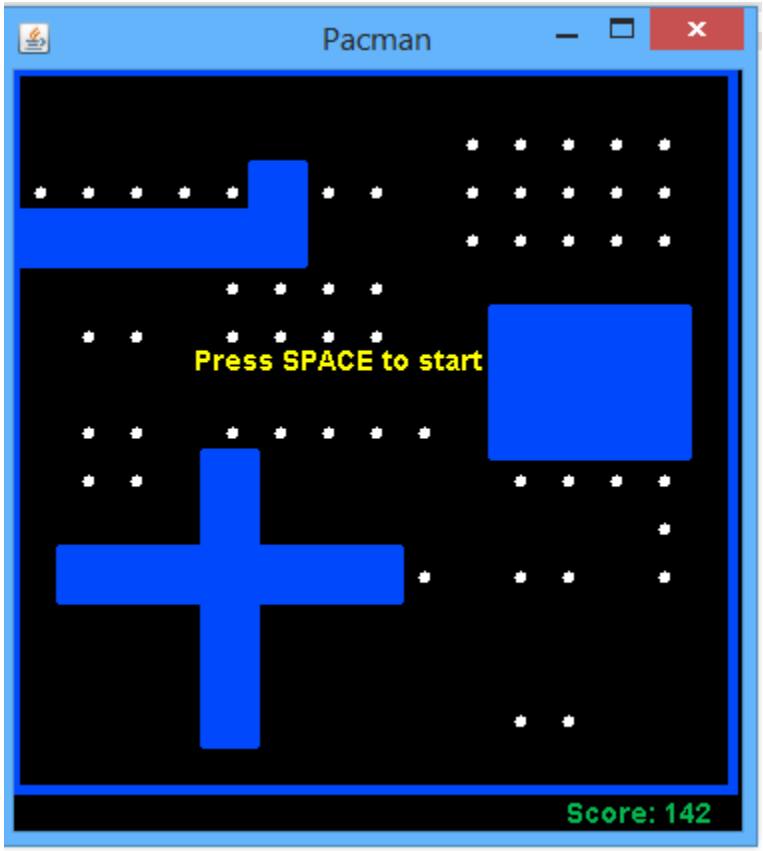



Pacman
Score: 1


Pacman
Score: 70


Pacman
Score: 84


Pacman
Score: 142


Pacman
Press SPACE to start
Score: 142