Date : / /
Page No.:

1. Program to draw basic graphic objects

* Procedure :

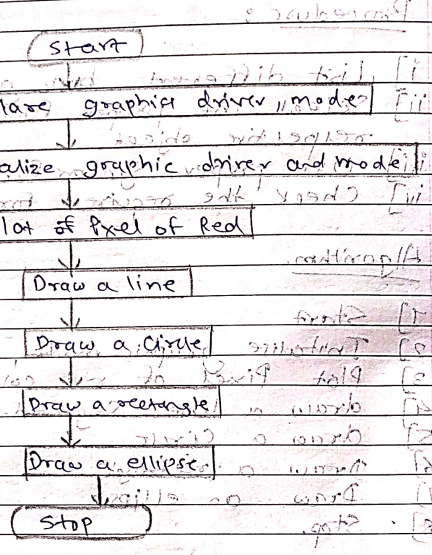
i] List different basic graphics function.
ii] Select the proper function to draw respective object.
iii] Use proper syntax for selected function
iv] Check the required format of output.

* Algorithm.

1] Start
2] Initialise graphic mode, graphic driver
3] Plot Pixel of red color
4] draw a line segment.
5] draw a circle
6] Draw a rectangle
7] Draw an ellipse
8] Stop.

Date : / /
Page No.:

Flowchart



Date : / /
Page No.:

C - Code

```
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
Void main()
{
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TC3\\BGI");

putpixel(50,50,RED);
outtextxy(55,55,"Pixel");

line(100,70,200,100);
outtextxy(220,110,"line");

Circle(230,50,50);
outtextxy(220,120,"Circle");

rectangle(300,40,400,100);
outtextxy(320,120,"Rectangle");
ellipse(450,50,0,360,100,75);
outtextxy(450,200,"Ellipse");
getch();
}
```

* Practical Related Questions:

1] Find the minimum and maximum coordinates of screen.

→
i] Use xmax to find maximum coordinate of x
ii] Use xmin to find minimum coordinate of x
iii] Use ymax to find maximum coordinate of y
iv] Use ymin to find minimum coordinate of y.

2] What is graphics driver and graphics mode?

→
i] The graphics driver is a program that controls how your graphic components work with the rest of your computer.
ii] While working in C programming, default output mode is text.
iii] To draw graphics objects on screen display mode must be changed to graphics mode.

3] what is path of graphics driver?

→
C:\\TC3\\BGI

4] Find error in following code:

```
{
putpixel(20,20);
outtextxy(25,25,"Pixel");
}
```

→ In putpixel: Didn't specified Color: Syntax error.

5] List four applications of Computer graphics.

→
i] Computer Art.
ii] Computer Aided Drawing
iii] Education.
iv] Visualisation

* Conclusion:

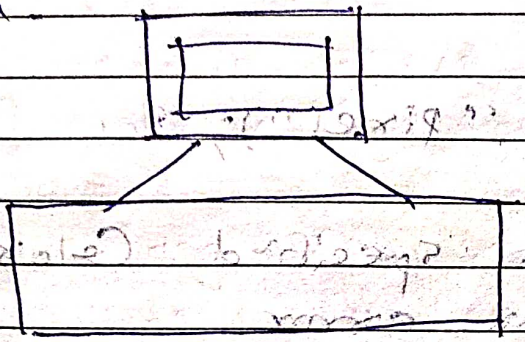
Computer graphics provides different graphic functions to draw various graphic objects.

Date : / /
Page No.:

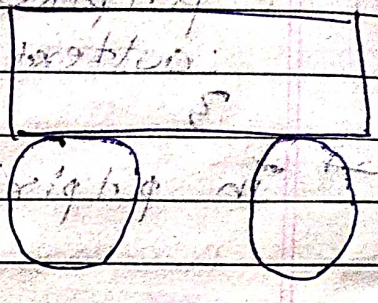
Exercise

1] Write a C program to display following objects.

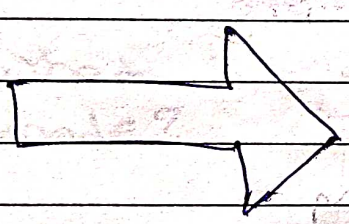
a.



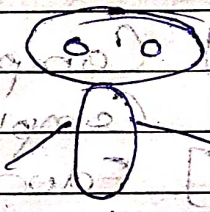
b.



c.

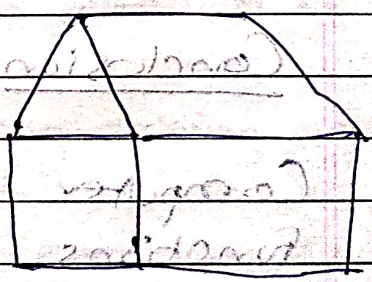


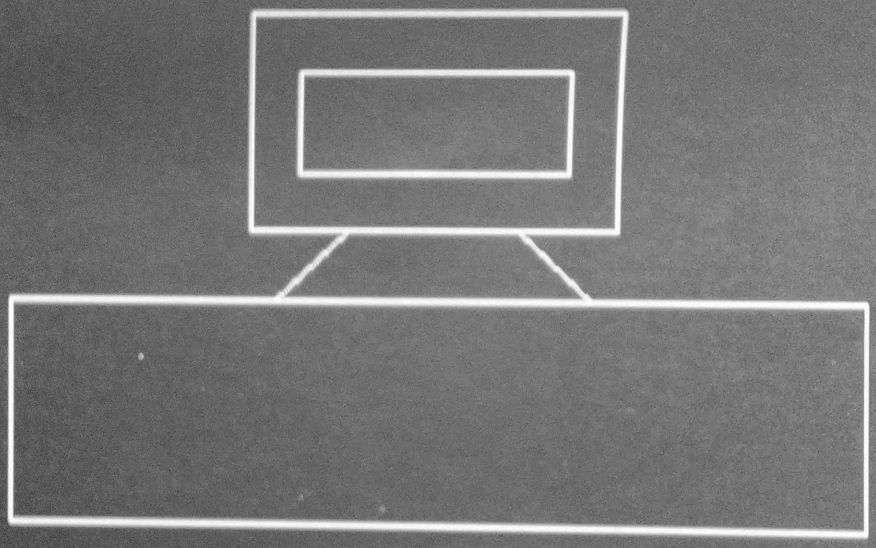
d.

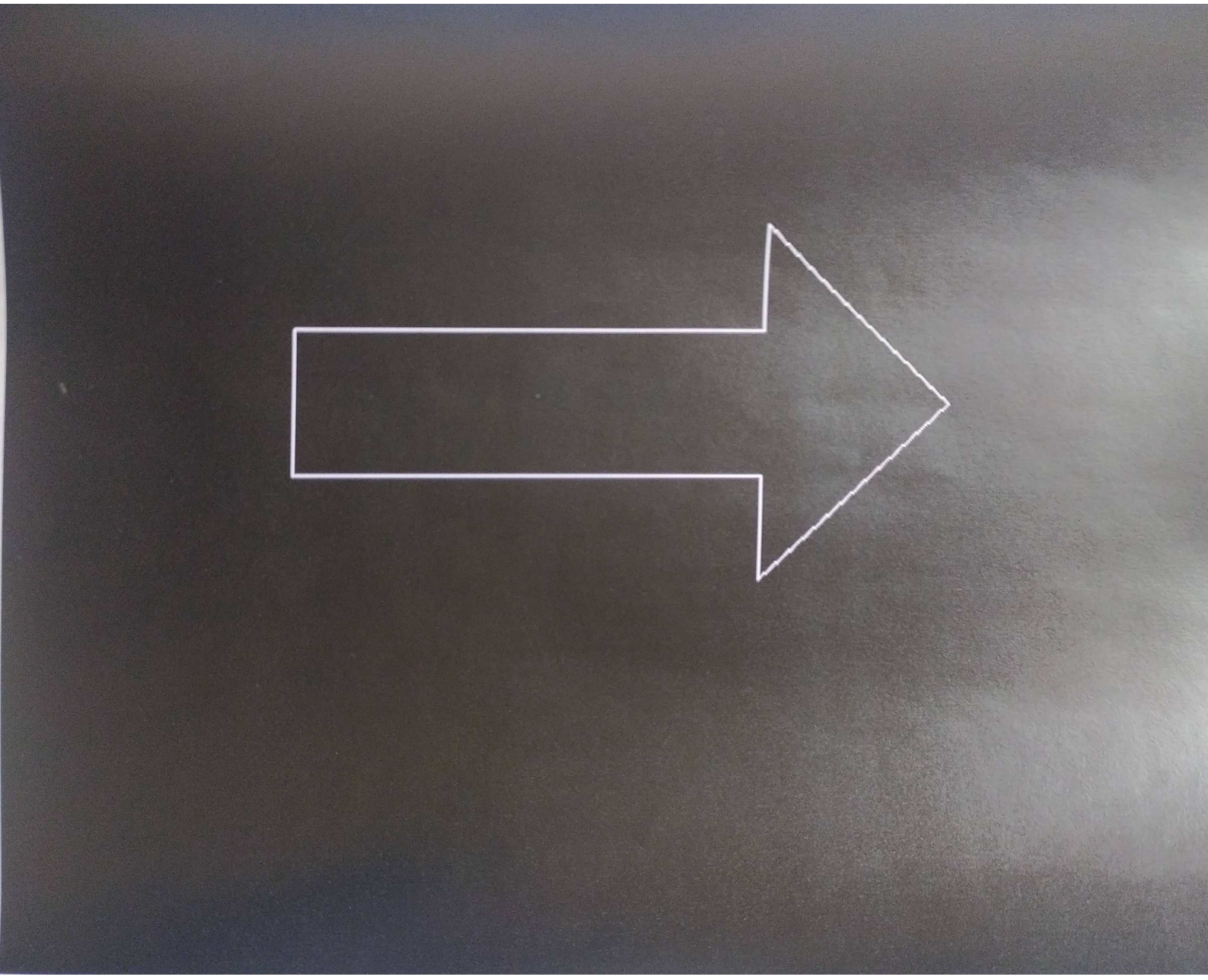


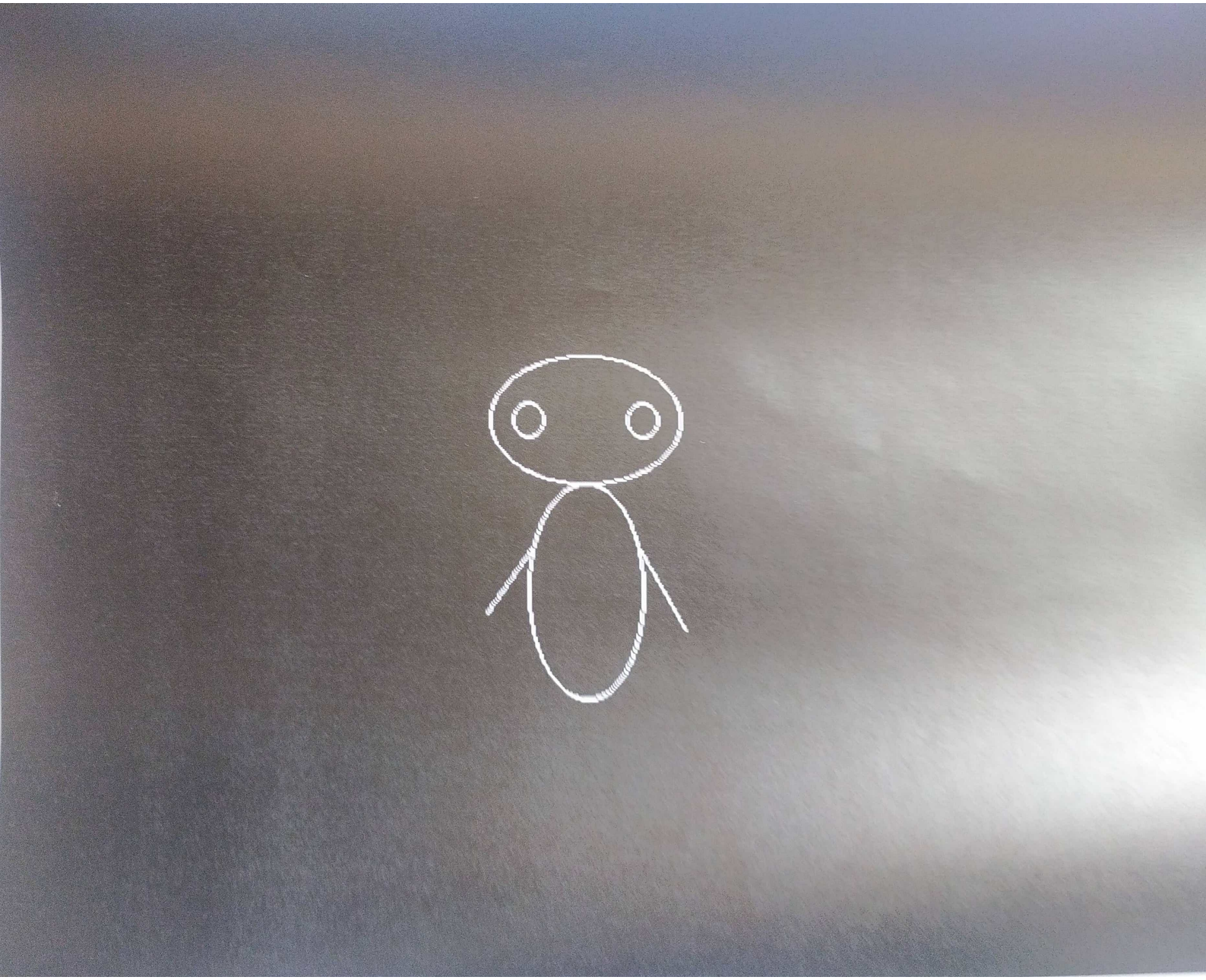
e.

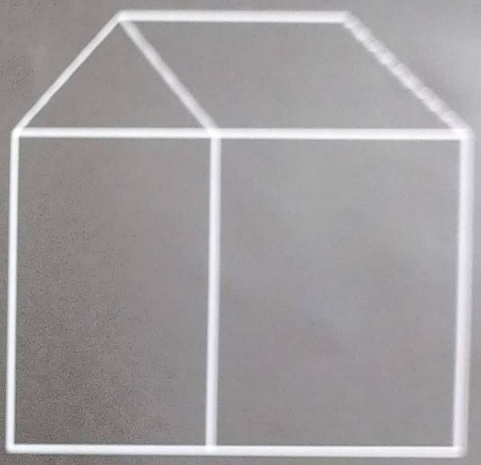
f.











Date : / /
Page No.:

2. Program to draw line using DDA algorithm

Procedure:

i] Read the input of the 2 end points of the line as $(x1, y1)$ & $(x2, y2)$ such that $x1 != x2$ and $y1 != y2$

ii] Calculate $dx = x2 - x1$ and $dy = y2 - y1$

iii] if $(dx >= dy)$
    step = dx
else
    step = dy

iv] $xin = dx / step$ & $yin = dy / step$

v] $x = x1 + 0.5$ & $y = y1 + 0.5$

vi] For (k=0; k<step; k++)
{
    x = x + xin
    y = y + yin
    put pixel (x, y)
}

Date : / /
Page No.:

7. Program to draw line using DDA algorithm

Procedure:

i] Read the input of the 2 end points of the line as $(x1, y1)$ & $(x2, y2)$ such that $x1 != x2$ and $y1 != y2$

ii] Calculate $dx = x2 - x1$ and $dy = y2 - y1$

iii] if $(dx >= dy)$
    step = dx
  else
    step = dy

iv] $xin = dx/step$ & $yin = dy/step$

v] $x = x1 + 0.5$ & $y = y1 + 0.5$

vi] for $(k = 0; k < step; k++)$
  {
    $x = x + xin$
    $y = y + yin$
    put pixel (x, y)
  }

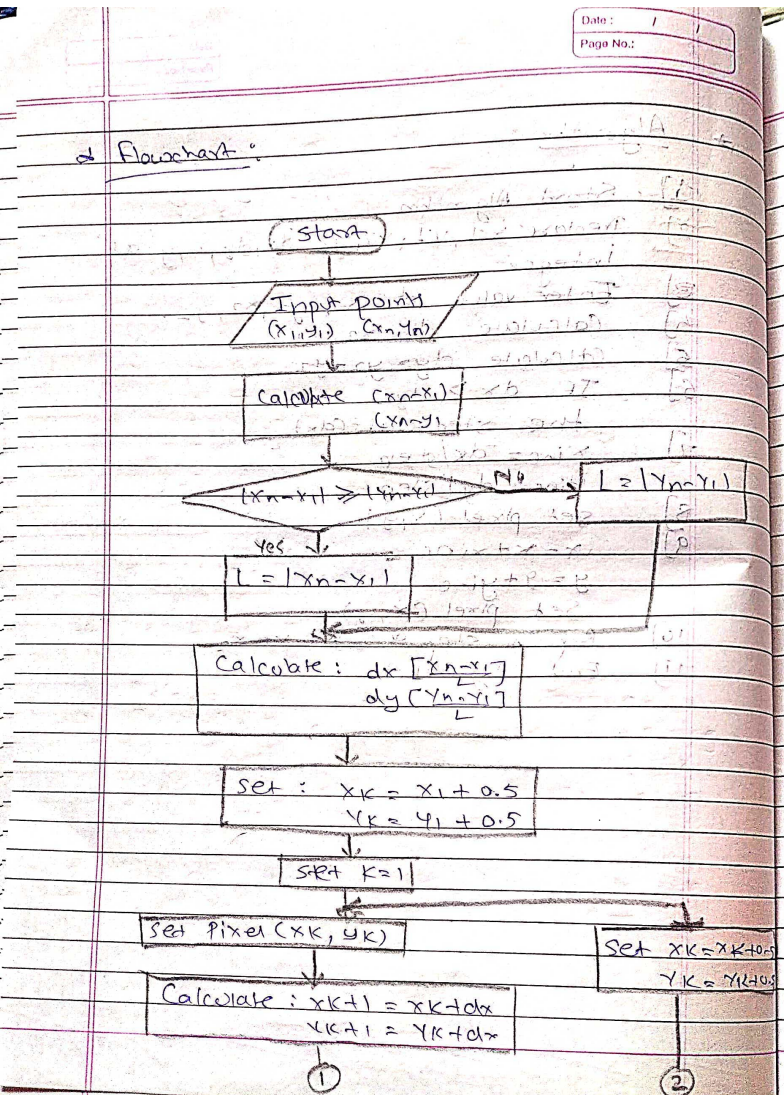
Date : / /
Page No.:

Algorithm:

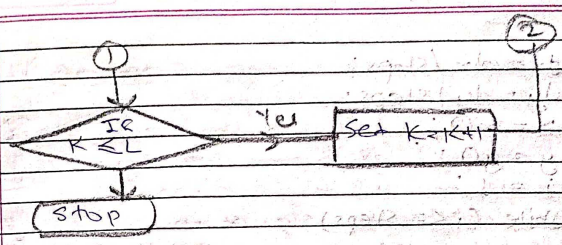
1] Start Algorithm
2] Declare x1, y1, x2, y2, dx, dy, x, y as integer
3] Enter value of x1, y1, x2, y2.
4] Calculate dx = x2 - x1
5] Calculate dy = y2 - y1
6] If dx > dy
  then step = abs(dx)
7] xinc = dx/step
  yinc = dy/step
8] Set pixel (x, y)
9] x = x + xinc
  y = y + yinc
  Set pixel (x, y)
10] Repeat step 9 until x = x2
11] End Algorithm.

Date :
Page No.:

Flowchart :



Date :
Page No.:



C - Code

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
void main()
{
  int gd= DETECT, gm, i;
  float x, y, dx, dy, steps;
  int x0, x1, y0, y1;
  initgraph (&gd, &gm, "C:\\TC\\BGI");
  setbkcolor(WHITE);
  x0=-100, y0=200, x1=500, y1=300;
  dx(float) (x1-x0);
  dy(float) (y1-y0);
  if (dx)=dy)
    {
      steps =dx;
    } else
        {
          steps=dy;
    }
```

```
dx = dx / steps;
dy = dy / steps;
x = x0;
y = y0;
i = 1;
while (i <= steps)
{
    putpixel (x, y, RED);
    x += dx;
    y += dy;
    i = i + 1;
}
getch();
closegraph();
}
```

* Practical Related Question

1] Define the term Rasterization.

→ Rasterization is the task of taking an image described in a vector graphics format and converting it into a raster image.

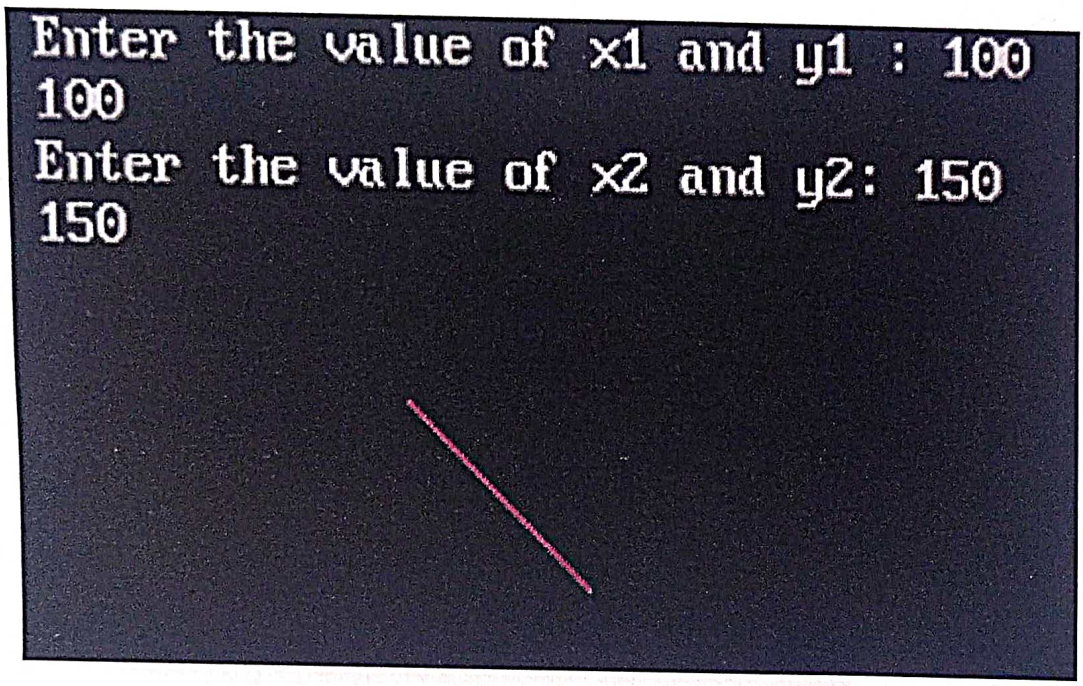
2] Write slope intercept form of a line

→ Slope intercept form of a line —

$$y = mx + b$$

m = slope of line

b = y intercept of line


Enter the value of x1 and y1 : 100
100
Enter the value of x2 and y2: 150
150

Date : / /
Page No.:

3] Write advantages and disadvantages of DDA Algorithm.

→

Advantages :

1] It is a simple Algorithm
2] It is easy to implement.
3] Avoid using multiplication which is costly in term of time complexity

Disadvantages :

1] Points generated are not accurate.
2] Resulted lines are not smooth
3] There is an increase in time complexity because of round off () function.

* Conclusion :

i) In Computer Graphics, a digital differential analyzer (DDA) is used for interpolation of variables over an interval between start and end point.
ii) DDA's are used for rasterization of lines, triangles and polygons

* Exercise

1] Give following values for every iteration of DDA algorithm to draw a line from (3,4) to (6,8).

→

$x1 = 3 \quad y1 = 4$

$x2 = 6 \quad y2 = 8$

$dx = x2 - x1$

$= 6 - 3$

$dx = 3$

$dy = y2 - y1$

$= 8 - 4$

$dy = 4$

$dy > dx$ so $l = dy$

$dx = x2 - x1 / l$

$= \frac{3}{4}$

$= 0.75$

$dy = y2 - y1 / l$

$= 4/4$

$= 1$

$x = 3 + 0.5 = 3.5$

$y = 4 + 0.5 = 4.5$

i=1

$x = 3.5 + 0.75$

$= 4.25$

$y = 4.5 + 1$

$= 5.5$

i=2

$x = 4.25 + 0.75$

$= 5$

$y = 5.5 + 1$

$= 6.5$

i=3

$x = 5 + 0.75$

$= 5.75$

$y = 6.5 + 1$

$= 7.5$

i=4

$x = 5.75 + 0.75$

$= 6.5$

$y = 7.5 + 1$

$= 8.5$

I	X	Y
1	4.25	5.5
2	5	6.5
3	5.75	7.5
4	6.5	8.5

ii) Give following values of every iteration of DDA Algorithm to draw a line From (-5, -5) to (-12, -12).

→

$x1 = -5$ $\quad y1 = -12$

$x2 = -12$ $\quad y2 = -12$

$dx = x_2 - x_1$

$= 5 - (-5)$

$= 10$

$dy = y_2 - y_1$

$= (-12) - (-12)$

$= (-12) + 12$

$= 0$

$dy < dx$ so $d = 10$

$dx = x_2 - x_1 / d$

$= \frac{10}{10}$

$= 1$

$dy = y_2 - y_1 / d$

$= 0/10$

$= 0$

$x = 10 + 0.5 = 10.5$

$y = 0 + 0.5 = 0.5$

i = 1

$x = 10.5 + 1 = 11.5$

$y = 0.5 + 0 = 0.5$

i = 2

$x = 11.5 + 1 = 12.5$

$y = 0.5 + 0 = 0.5$

i = 3

$x = 12.5 + 1 = 13.5$

$y = 0.5 + 0 = 0.5$

i = 4

$x = 13.5 + 1$

$= 14.5$

$y = 0.5 + 0 = 0.5$

I	x	Y
1	11.5	0.5
2	12.5	0.5
3	13.5	0.5
4	14.5	0.5

Date : / /
Page No.:

3. Program to draw line using Bresenham's algorithm.

Procedure:

i] Input the two line end-point, storing the left end-point (x_0, y_0)

ii] Plot the point (x_0, y_0)

iii] Calculate the constants Δx, Δy, $2\Delta y$ and $2(\Delta y - 2\Delta x)$ and get the first value for decision parameter at $P_0 = 2\Delta y - 2\Delta x$

iv] At each x_k along the line, starting at $k=0$, perform following test:
If $p_k < 0$ then next point (x_{k+1}, y_k) and $p_{k+1} = p_k + 2\Delta y$
Otherwise, the next point to plot is (x_{k+1}, y_{k+1}) and $p_{k+1} = p_k + 2\Delta y - 2\Delta x$

v] Repeat step 4 (Δx) times

Algorithm:

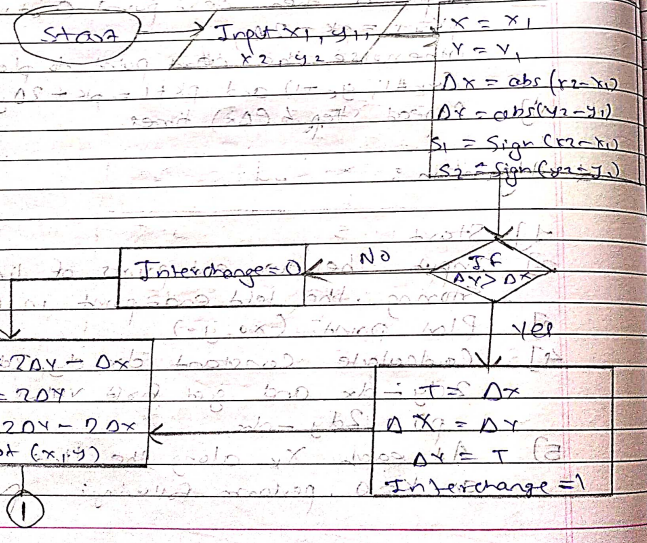
1] Start
2] Input the two end-points of line, storing the left end-point in (x_0, y_0)
3] Plot point (x_0, y_0)
4] Calculate constant dx, dy, 2dy and 2dy - dx and get first value for $p_0 = 2dy - dx$
5] At each x_k along the line, starting at $k=0$, perform following:

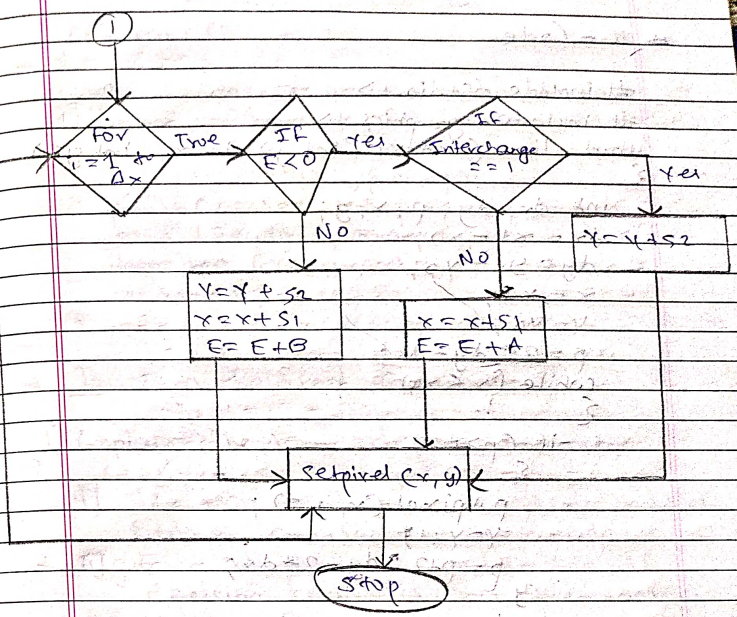
If $p_k < 0$ next point is (x_{k+1}, y_k)
and
$p_{k+1} = p_k + 2dy$

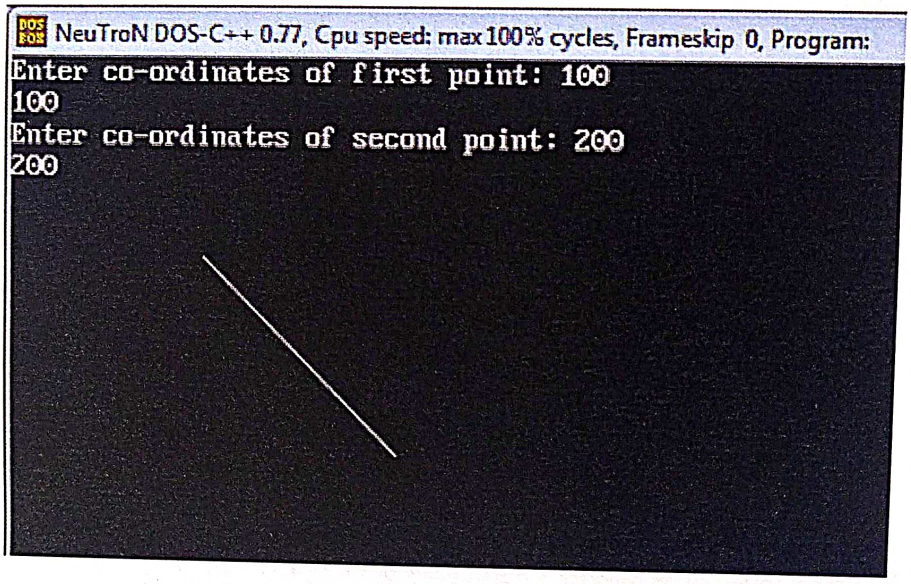
Otherwise,
(x_k, y_{k+1})
$p_{k+1} = p_k + 2dy - 2dx$

5] Repeat step 4 $dx-1$ times
6] End

* Flowchart :






NeuTroN DOS-C++ 0.77, Cpu speed: max 100% cycles, Frameskip 0, Program:
Enter co-ordinates of first point: 100
100
Enter co-ordinates of second point: 200
200

Date: / /
Page No.:

C - Code

```
#include <stdio.h>
#include <graphics.h>
void line(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)
  {
    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;
  }
}
```

Date: / /
Page No.:

```
int main()
{
  int gd = DETECT, gmode = error, x0, y0, x1, y1;
  initgraph(&gd, &gmode, "C:\\TC\\BGI");
  printf("Enter 1st point ");
  scanf("%d%d", &x0, &y0);
  printf("Enter 2nd point ");
  scanf("%d%d", &x1, &y1);
  line(x0, y0, x1, y1);
  return 0;
}
```

Practical Related Questions:

1] Explain the term decision parameter.

→
i] It is a criteria based on which further calculations are made.
ii] Ex. in bresenham algorithm if decision factor is ≥ 0 then action taken place else not.

2] Write advantages of Bresenham's algorithm over DDA.

→ i] Easy to Implement
ii] Fast and incremental
iii] More Accurate points than DDA
iv] Uses Fixed points only.

* Exercise

i) Give following values for every iteration of Breisenham's algorithm to draw from (3,4) to (6,8)

→

$x_1 = 3$ $y_1 = 4$

$x_2 = 6$ $y_2 = 8$

$\Delta x = 3$ $\therefore 2\Delta x = 6$

$\Delta y = 4$ $\therefore 2\Delta y = 8$

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{8-4}{6-3} = \frac{4}{3} = 1.3$$

$pk = 8 - 3 = 5$

	x_k, y_k	p_k	x_{k+1}, y_{k+1}
	(3,4)	(5)	(4,5)

$p_{k+1} = 2 + 8 - 6\,(5-4)$

$= 4$

	x_k, y_k	p_k	x_{k+1}, y_{k+1}
①	(4,5)	4	(5,6)

$P_{k+1} = 10 + 6 - 8\,(5-4)$

$= -2$

②	(5,6)	-2	(5,7)

$P_{k+1} = -2 + 6 - 8\,(5-5)$

$= 4$

③	(5,7)	4	(6,8)

ii) Give following values for every iteration of Breisenham's algorithm to draw from (-6,-6) to (-14,-14).

→

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{-14 - (-6)}{-14 - (-6)} = \frac{-8}{-8} = 1$$

$\Delta x = -8$ $\Delta y = -8$

$\Delta y = -16$ $2\Delta y = -16$

$\therefore pk = 2\Delta x - \Delta y$

$= (-16) - (-6)$

$pk = -8$

⓪	(-6,-6)	-8	(-6,-5)

$p_{k+1} = -8 + (-16) - (-16)\,(-6 - (-6))$

$= -24$

①	(-6,-5)	-24	(-6,-4)

$p_{k+1} = -24 + (-16) - (-16)\,(-6 - (-6))$

$= -40$

②	(-6,-4)	-40	(-6,-3)

$p_{k+1} = -40 + (-16) - (-16)\,(-6 - (-6))$

$= -56$

③	(-6,-3)	-56	(-6,-2)

$p_{k+1} = -56 + (-16) - (-16)\,(-6 - (-6))$

$= -72$

④ (-6, -2) -72 (-6, -1)

$p_{k+1} = -72 + (-16) - (-10)(-6 - (-6))$

$= -88$

⑤ (-6, -1) -88 (-6, 0)

$p_{k+1} = -88 + (-16) - (-16)(-6 - (-6))$

$= -104$

⑥ (-6, 0) -104 (-5, 1)

$p_{k+1} = -104 + (-16) - (-18)(-6 - (-6))$

$= -120$

⑦ (-6, 1) -120 (-6, 2)

$p_{k+1} = -120 + (-16) - (-16)(-6 - (-6))$

$= -136$

* Assessment Scheme

Omkar Savant.
Atharva Raut.

Page No.:

4. Program to draw Circle

Date: / /
Page No.:

* Procedure:

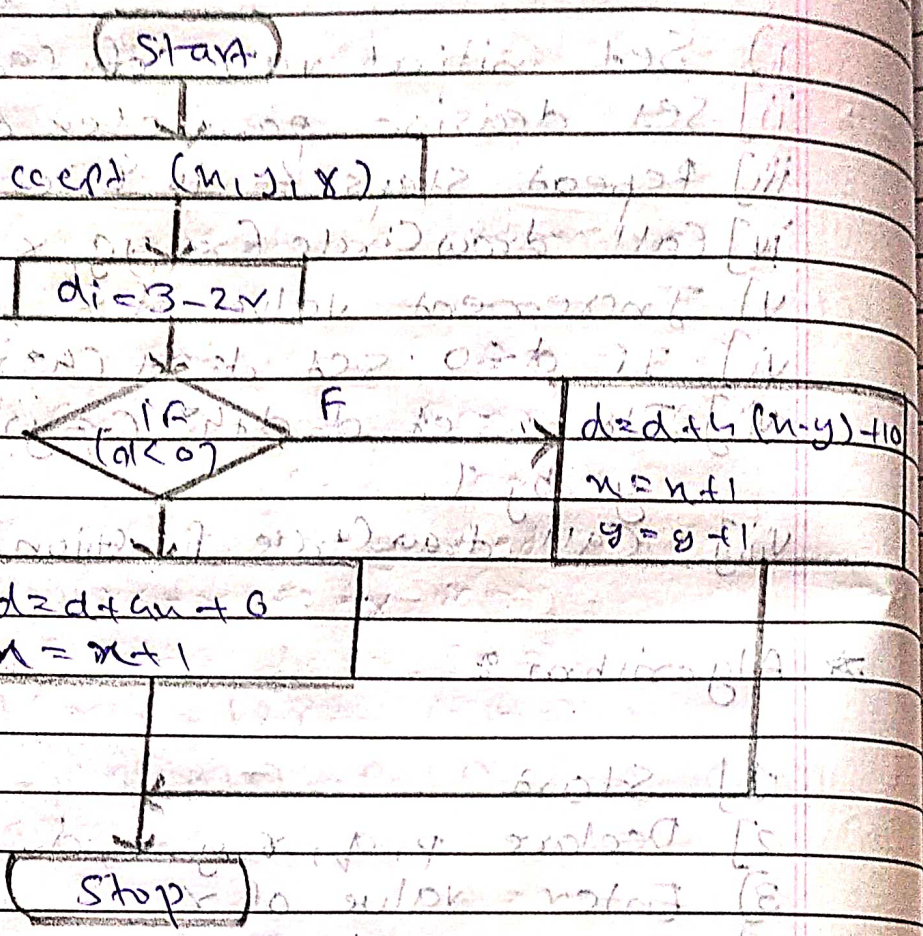
i) Set initial value of center of circle
ii) Set decision parameter d to $d = 3 - (2*r)$
iii) Repeat steps 4 to 8 until $x <= y$
iv) Call drawCircle (xc, yc, x, y) function
v) Increment value of x
vi) If $d < 0$, set $d = d + (4 * x) + 6$
vii) Else, set $d = d + 4 * (x - y) + 10$ and decrement y by 1.
viii) Call drawCircle function.

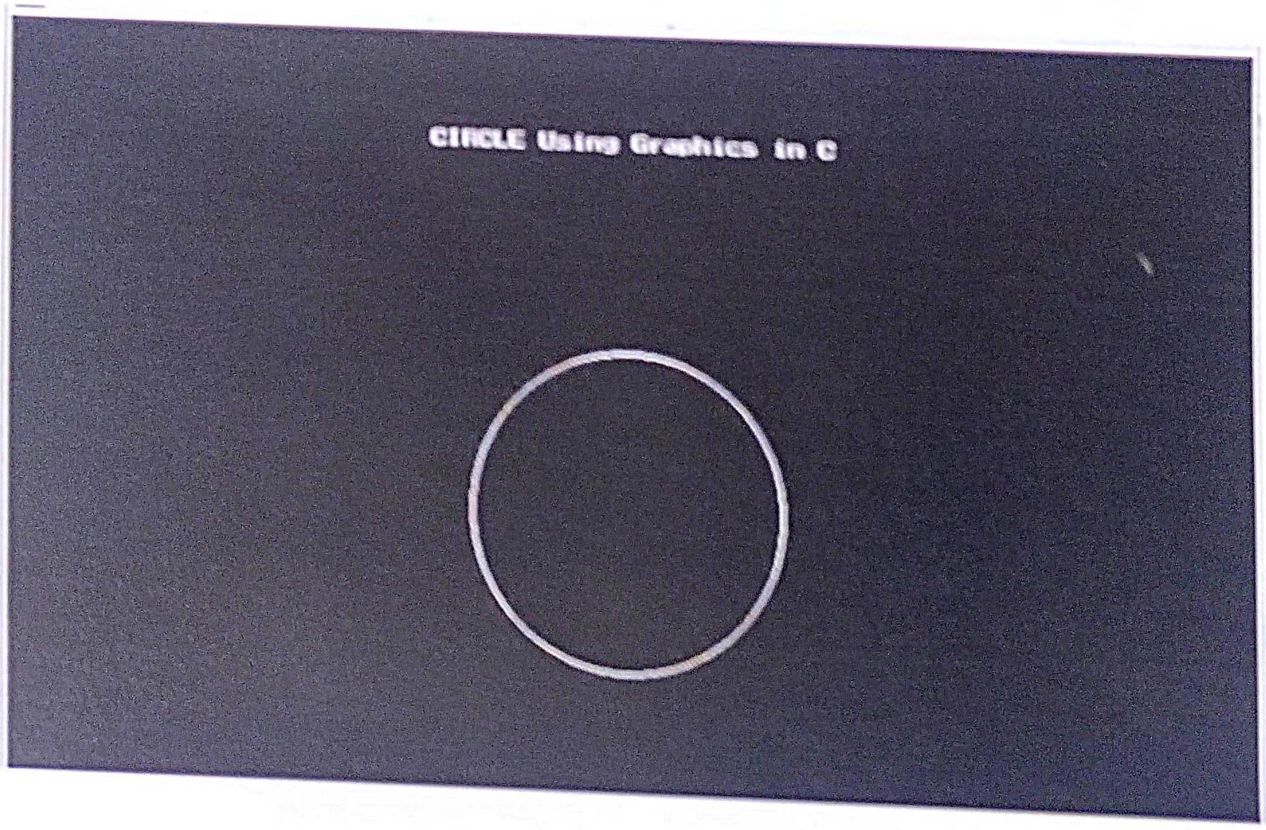
* Algorithm:

1] Start
2] Declare p, q, s, y, r, d variable
3] Enter value of r
4] Calculate $d = 3 - 2r$
5] Initialize $x = 0$
6] Check if whole circle is scan converted
if $x >= y$
Stop
7] Plot eight-points by eight-way symmetry
8] Find location of next pixel to be scanned
9] Go to step 6
10] Stop

Date : / /
Page No.:

* Flowchart




CIRCLE Using Graphics in C

Date : / /
Page No.:

C - Code :

```
#include <graphics.h>
#include <stdio.h>
void pixel(int xc, int yc, int x, int y);
int main()
{
    int gd, gm, xc, yc, r, x, y, p;
    initgraph(&gd, &gm, "C:\\TC\\BGI");
    printf("Enter Center of circle:");
    scanf("%d%d", &xc, &yc);
    printf("Enter radius of circle:");
    scanf("%d", &r);
    x = 0;
    y = r;
    p = 3 - 2 * r;
    pixel(xc, yc, x, y);
    while (x < y)
    {
        if (p < 0)
        {
            x++;
            p = p + 4 * x + 6;
        } else
        {
            x++;
            y--;
            p = p + 4 * (x - y) + 10;
        }
        pixel(xc, yc, x, y);
```

```
getch();
closegraph();
return 0;
}
void pixel(int xc, int yc, int x, int y)
{
putpixel(xc+x, yc+y, Red);
putpixel(xc+y, yc+x, 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);
}
```

* Practical Related Questions:

1] Write equation of Circle.

→ The equation of circle is $x^2 + y^2 = r^2$

2] Write algorithm to draw 8-way symmetry of Circle.

→

1] Start
2] Declare p, q, x, y, r, d variable
3] Enter value of r
4] Calculate $d = 3 - 2r$
5] Initialize $x = 0$
6] Check if whole circle is scan converted
7] Plot eight point by using concepts of eight-way symmetry.
8] Find location of next pixel to be scanned.
9] Go to Step 6.
10] Stop

3] How the value of decision parameter (d) is calculated.

→ $d = 3 - 2r$

where,

d = decision parameter
r = radius.

* Conclusion:

There are two popular algorithm for generating a circle - Bresenham's Algorithm and Midpoint Circle Algorithm.

These Algorithm are based on the idea of determining the subsequent points required to draw the circle.

Date :
Page No.:

Exercise

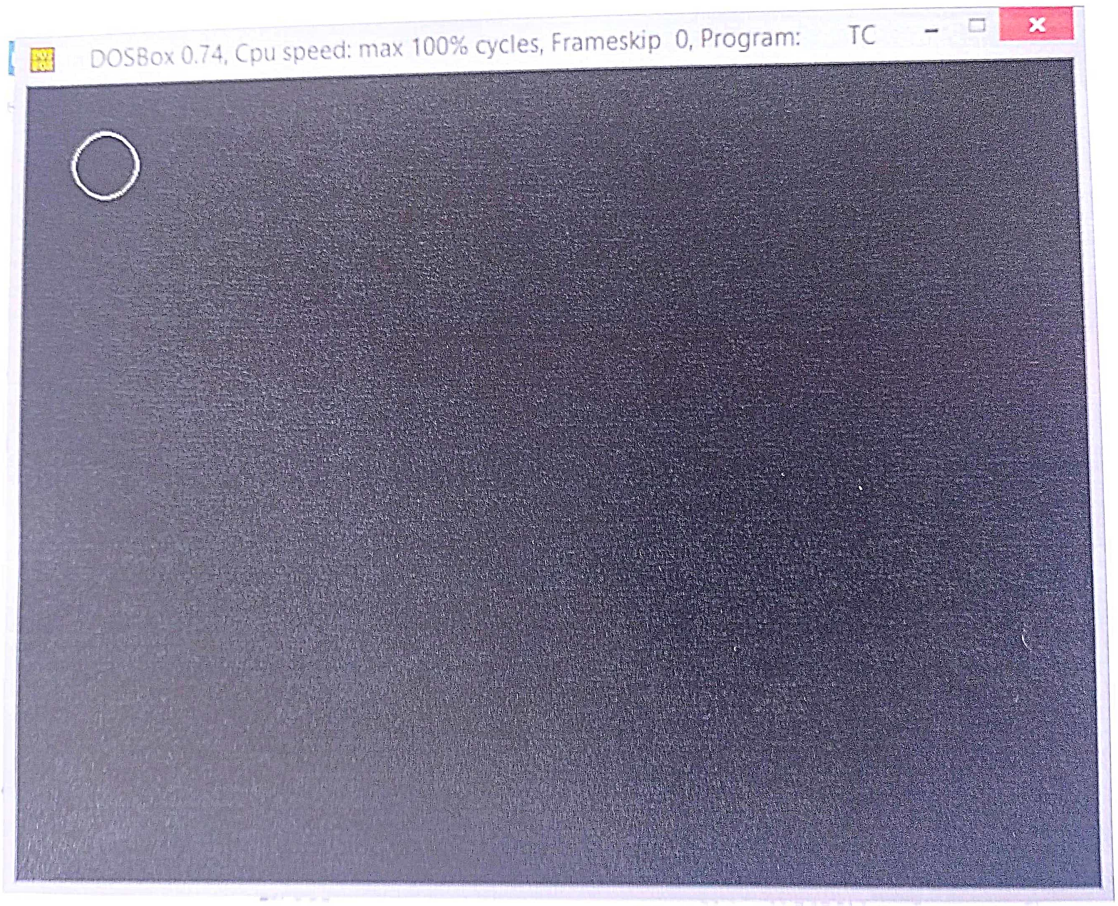
1] Calculate pixel for circle with radius 10 using Bresenham's Algorithm.

→

P_K	P_{K+1}	(x_{K+1}, y_{K+1})	(xplot, yplot)
		(0, 10)	(10, 20)
-17	-7	(1, 10)	(11, 20)
-7	7	(2, 10)	(12, 20)
7	-7	(3, 9)	(13, 19)
-7	15	(4, 9)	(14, 19)
15	13	(5, 8)	(15, 18)
13	19	(6, 7)	(16, 17)

2] Draw a circle with center (50, 50) and radius 20 by using Breesenham's algorithm.

→


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

5. Program to fill polygon.

Date : / /
Page No.:

* Procedure :

1] If target-color is equal to replacement color, return.
2] If the color of node is not equal to target-color, return.
3] Set the color of node to replacement-color.
4] Perform Flood-Fill (South of node, color, re-color)
Flood Fill (North of node, color, re-color)
Flood Fill (East of node, color, re-color)
Flood Fill (West of node, color, re-color)
5] Return.

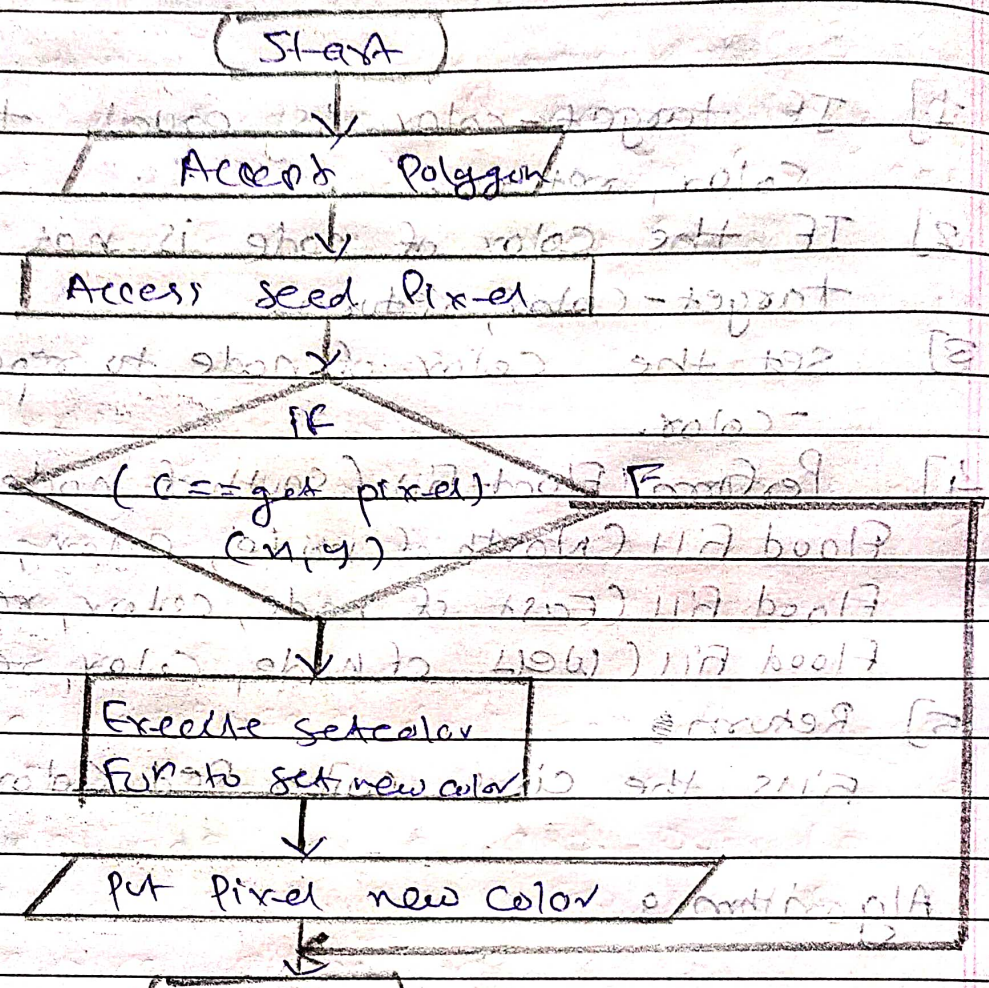
Fills the circle with Red Color.

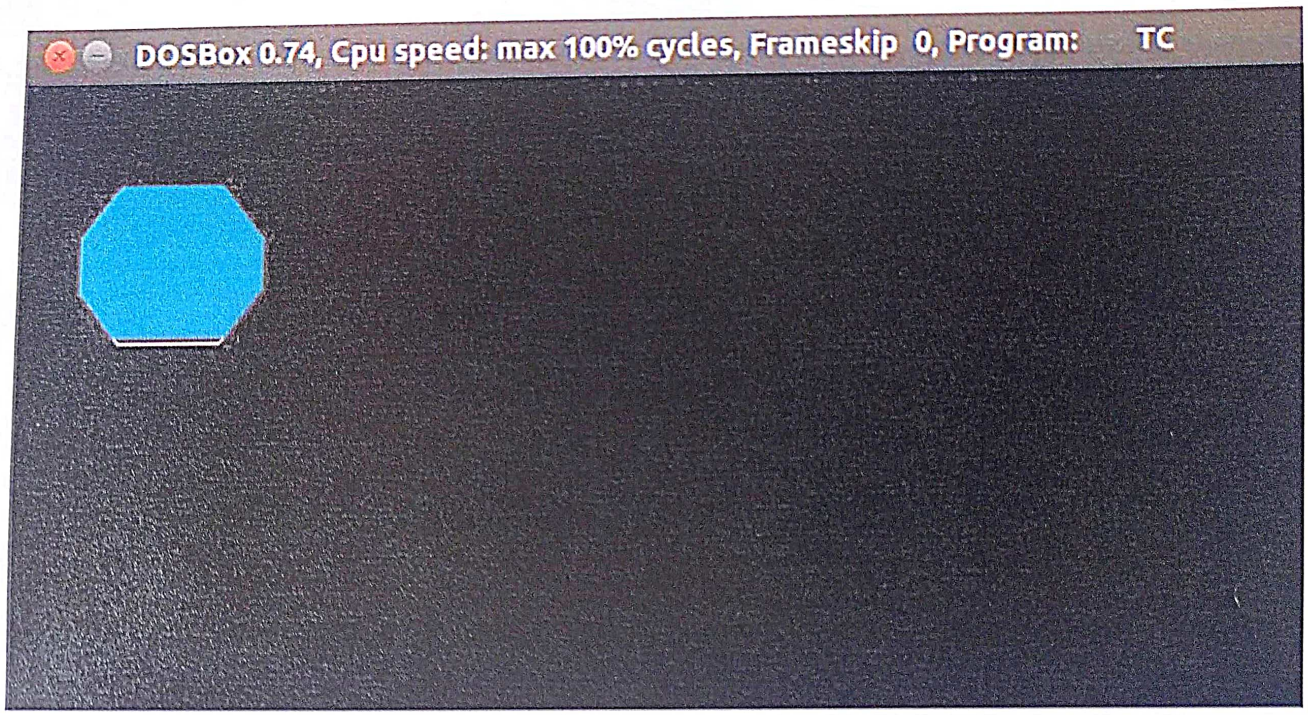
* Algorithm :

1] Initialize the value of seed point
2] Define the boundary values of polygon.
3] Check if the current seed point is of default color, then repeat steps 4 and 5 till the boundary pixels reached.
4] Change the default color with the fill color at the seed point.
5] Recursively follow the procedure with four neighbourhood points.
6] Exit.

Date :
Page No.:

* Flowchart




DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

Date : / /
Page No.:

* Practical Related Questions :

1) Define polygon.

→ A polygon is a plane figure with at least three straight sides and angles.

2) Explain types of polygon.

→

i) Regular Polygon :

A regular polygon has equal length sides with equal angles between each side.

ii) Irregular Polygon :

It has unequal length sides and unequal angles between sides.

3) List coordinates of neighboring pixels in 8-connected method for seed pixel with coordinates (x, y).

→

```
Putpixel (x, y, color);
Fill (x-1, y, color);
Fill (x+1, y, color);
Fill (x, y-1, color);
Fill (x, y+1, color);
Fill (x+1, y+1, color);
Fill (x-1, y+1, color);
```

```
Fill(x-1, y-1, color);
Fill(x+1, y-1, color);
```

4] List coordinates of neighbouring pixels in 4-connected method for seed pixel with co-ordinates (x,y).

→

```
Putpixel(x, y, color);
Fill(x+1, y, color);
Fill(x, y+1, color);
Fill(x-1, y, color);
Fill(x, y-1, color);
```

5] Explain inside-outside test of polygon.

→ The Inside-outside test is used to find the inside and outside region of a polygon. Boundary fill is a recursive algorithm.

★ ~~Conclusion~~ Exercise.

1] WAP to draw hexagon and fill hexagon with pink color using Flood fill algorithm with 8-connected method.

→

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

```
Void fill(int x, int y, int old, int new)
{
    int current;
    current = getpixel(x,y);
    if(current == old)
    {
        delay(5);
        putpixel(x, y, new);
        fill(x+1, y, old, new);
        fill(x-1, y, old, new);
        fill(x, y+1, old, new);
        fill(x, y-1, old, new);
        fill(x+1, y+1, old, new);
        fill(x-1, y-1, old, new);
        fill(x+1, y-1, old, new);
        fill(x-1, y+1, old, new);
    }
}
Void main()
{
    int gd= DETECT, gm;
    initgraph(&gd, &gm, "C:\\TC3\\BGI");
    int arr[]={300, 150, 400, 250, 350, 320};

    drawpoly(6, arr);
    floodfill(370, 70, 0, 15);
    getch();
    closegraph();
}
```

5] WAP to draw a triangle and fill it with Blue color using Flood fill algorithm with 4-connected method.

→

```
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
void fill(int, int, int, int);
void main()
{
  int gd=DETECT, gm;
  initgraph(&gd, &gm, "C:\\TC\\BGI");
  Triangle(120, 200, 120);
  flood(55, 55, 10, 0);
  getch();
}
void fill(int x, int y, int new, int def)
{
  if(getpixel(x,y)==def)
  {
    delay(1);
    putpixel(x, y, new);
    fill(x+1, y, new, def);
    fill(x, y+1, new, def);
    fill(x-1, y, new, def);
    fill(x, y-1, new, def);
  }
}
```

* Conclusion:

Polygon is a chain of connected line segments.
For filling polygons with particular colors, you need to determine the pixels falling on the border of the polygon and those which fall inside polygon.

* Assessment Scheme.

Omkar Sarant
Atharva Raut.

Date : / /
Page No.:

6. Program to Fill Polygon using boundary fill algorithm.

★ Procedure :

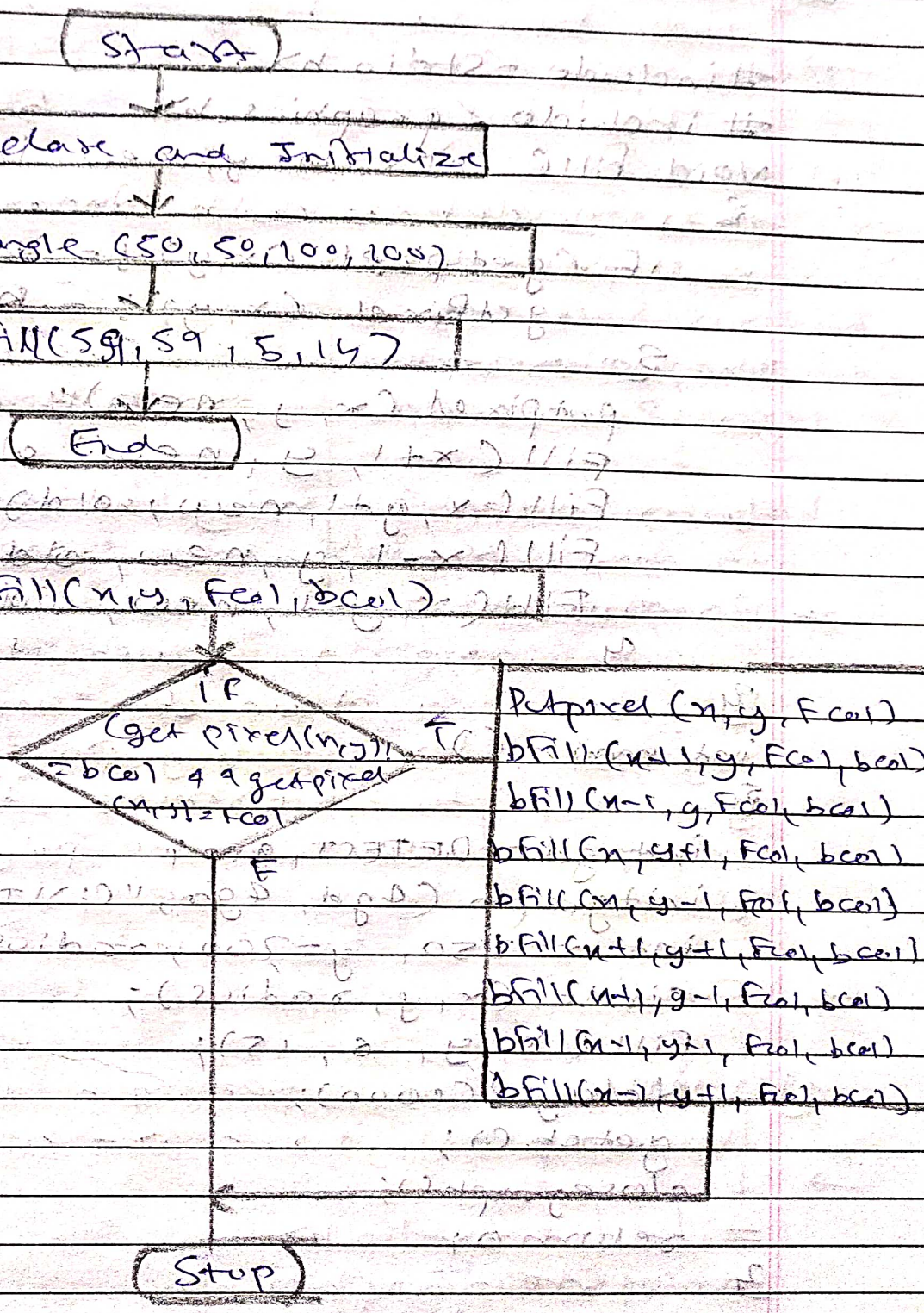
1] Initialize the value of seed point.
2] Define the boundary values of polygon
3] Check if the current seed point is of default color, then repeat the steps 4 and 5 till the boundary Pixel reached.
4] Change the default color with the fill color at seed point.
5] Recursively follow the procedure with four neighbourhood points.
6] Exit.

★ Algorithm :

1] Initialize the value of seed point.
2] Define the boundary value of polygon.
3] Check if the current seed point is of default color, then repeat the steps 4 and 5 till boundary fill is reached.
4] Change the default color with the fill color at seed point.
5] Recursively follow the procedure with four neighbourhood points.
6] Exit.

Date : / /
Page No.:

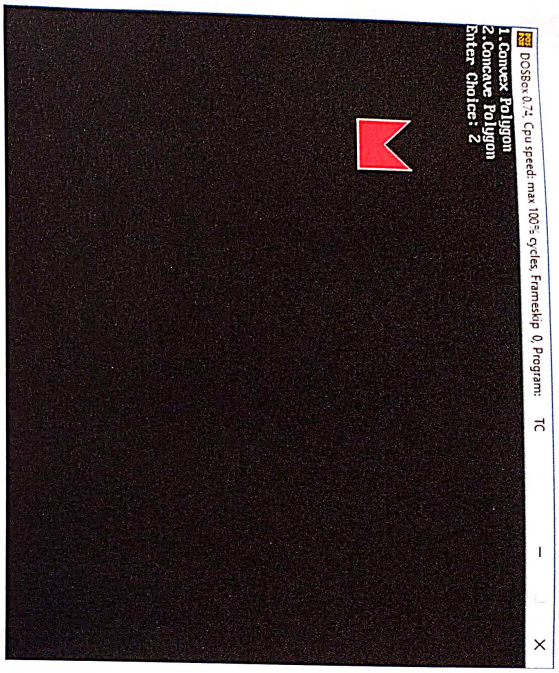
* Flowchart :



Page No.

* Code :

```
#include <stdio.h>
#include <graphics.h>
Void Fill( int x, int y, new, int old )
{
    if (getpixel (x,y) != old &&
        getPixel (x,y) != new)
    {
     putpixel (x, y, new);
     Fill ( x+1, y, new, old);
     Fill ( x, y+1, new, old);
     Fill ( x-1, y, new, old);
     Fill ( x, y-1, new, old);
    }
}
int main ()
{
 int gd = DETECT, gm;
 initgraph (&gd, &gm, "C:\TC3\BGI");
 int x=250, y=200, radius = 50;
 circle (x, y, radius);
 Fill ( x, y, 6, 15);
 delay (1000);
 getch ();
 closegraph();
 return 0;
}
```



Date : / /
Page No.:

Practical Related Questions :

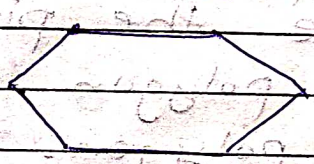
1] Compare 4-Connected and 8-Connected method to Fill polygon.

→

4-Connected	8-Connected
4 Connected method, is where a pixel may have upto four neighbour.	8 Connected method, is where a pixel may have upto 8 neighbours.
The pixels do not define a region since severed pixels such as A and B are not connected.	Using 8-connected definition we can identify a specific region

2] Identify type of polygon in following :

i]

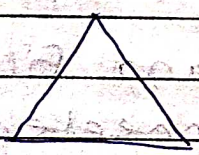


→ Hexagon
- Regular

ii]

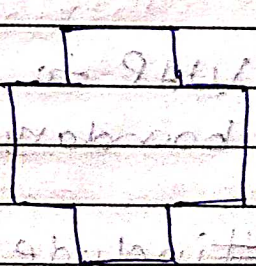
→ Irregular

iii]



→ Regular

iv]



→ Irregular

3] Give basic difference between Flood fill and boundary fill.

→

Flood Fill	Boundary Fill
Area is defined with multiple color.	Area is defined with single color.
The interior is coloured with any colour.	The interior is replaced with new color.
It is time Consuming	It is less time consuming

* Conclusion:

Polygon is a chain of connected line segment. For filling polygon with color, you need to determine the pixels falling on the border of the polygon and those which fall inside the polygon.

* Exercise.

1] WAP to draw pentagon and fill color using boundary fill with 8-connected method.

→

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

3] Give basic difference between flood fill and boundary fill.

→

Flood fill	Boundary fill
Area is defined with multiple color.	Area is defined with single color.
The interior is colored with any color.	The interior is replaced with new color.
It is time consuming	It is less time consuming

* Conclusion:

Polygon is a chain of connected line segment. For filling polygon with color, you need to determine the pixels falling on the border of the polygon and those which fall inside the polygon.

* Exercise.

1] WAP to draw pentagon and fill color using boundary fill with 8-connected method.

→

```
#include <stdio.h>
#include <graphics.h>
void fill(int x, int y, int new, int old)
{
  if (getpixel(x,y) != old &&
      getpixel(x,y) != new)
  {
    putpixel(x,y,new);
    fill(x+1,y,new,old);
    fill(x,y+1,new,old);
    fill(x-1,y,new,old);
    fill(x,y-1,new,old);
    fill(x+1,y-1,new,old);
    fill(x-1,y+1,new,old);
    fill(x-1,y-1,new,old);
    fill(x+1,y+1,new,old);
  }
}
int main()
{
  int gd=DETECT, gm;
  initgraph(&gd,&gm,"C:\\TC\\bgi");
  int arr[] = {120,100,240,220,120};
  drawpoly(5,arr);
  fill(5,5,4,15);
  delay(10000);
  getch();
  closegraph();
  return 0;
}
```

2] Write a program to draw trapezoid and fill it using boundary fill with 4-connected method.

→

```
#include <stdio.h>
#include <graphics.h>
void fill(int x, int y, int new, int old)
{
  if(getpixel(x,y) != old && getpixel(x,y) != new)
  {
    putpixel(x,y,new);
    fill(x+1,y,new,old);
    fill(x,y+1,new,old);
    fill(x-1,y,new,old);
    fill(x,y-1,new,old);
  }
}
int main()
{
  int gd=DETECT, gm;
  initgraph(&gd,&gm,"C:\\TC\\bgi");
  int arr=[100,120,350,240,100]
  drawpoly(5,arr);
  fill(x,y,6,15);
  delay(10000);
  getch();
  closegraph();
  return 0;
}
```

Date: / /
Page No.:

7. Program for two dimensional transformations (translation and scaling)

* Procedure:

1] Start
2] Input object co-ordinate
3] For Translation
   a] Enter translation factor
   b] Move the original co-ordinate to a new position.
   c] Display the object after translation.
4] For Scaling
   a] Input the scaled factor
   b] Transform the co-ordinate
   c] Display the object after scaling
5] Stop.

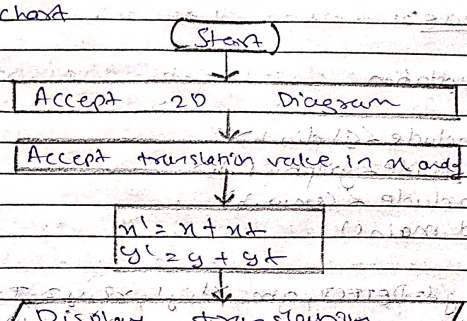
* Algorithm.

1] Start
2] Input Object coordinate
3] For Translation:
   a] Enter translation Factor
   b] Move original to new position
   c] Display translated object.
4] for scaling:
   a] Input scaling factor
   b] scale the coordinate
   c] Display scaled object
5] Stop.

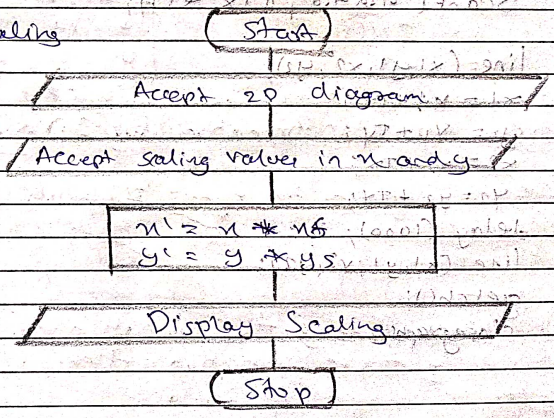
Date: / /
Page No.:

* Flowchart

1]



2] Scaling



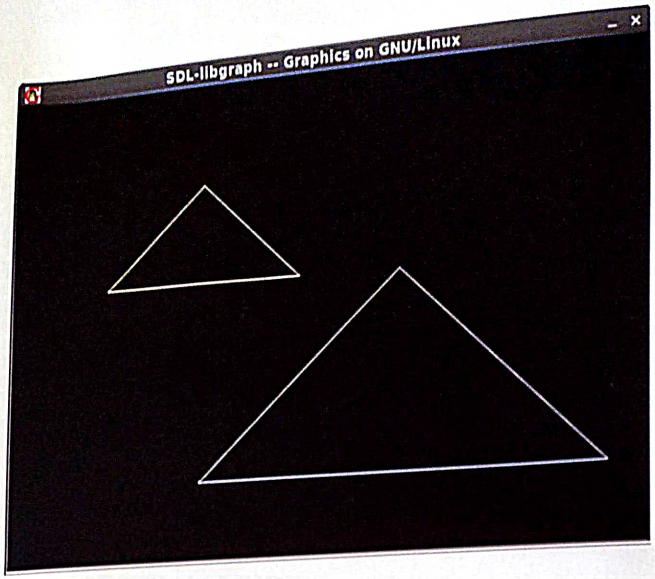
* Code:

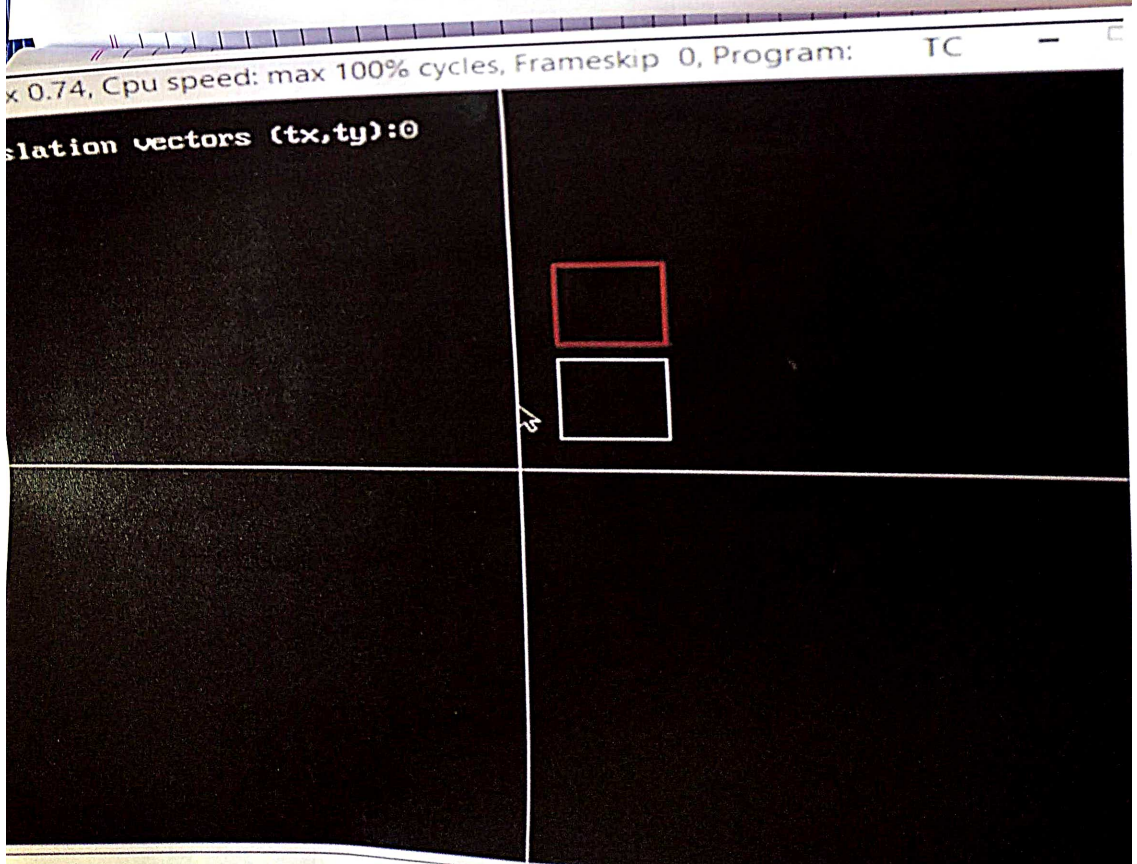
i] Translation

```
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
void main()
{
int gd=DETECT, gm, x1, y1, x2, y2, Tx, Ty;
initgraph(&gd, &gm, "C:\\TC3\\bgi");
printf("Enter line Coordinate:");
Scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
printf("Enter Translation Value:");
Scanf("%d%d", &Tx, &Ty);

line (x1, y1, x2, y2);
x1 = x1 + Tx;
y1 = y1 + Ty;
x2 = x2 + Tx;
y2 = y2 + Ty;
delay (1000);
line (x1, y1, x2, y2);
getch();
closegraph();
}
```

ii] Scaling

```
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
void main()
{
int gd= DETECT, gm, x1, y1, x2, y2;
int x, y;
initgraph(&gd, &gm, "C:\\TC3\\bgi");
clrscr();
printf("Enter Coordinates:");
Scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
printf("Enter Scaling Value:");
Scanf("%d%d", &x, &y);
rectangle (x1, y1, x2, y2);
x1 = x1 * x;
y1 = y1 * y;
x2 = x2 * x;
y2 = y2 * x;
delay(1000);
rectangle (x1, y1, x2, y2);
getch();
closegraph();
}
```


SDL-libgraph -- Graphics on GNU/Linux


x 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program:
TC
slation vectors (tx,ty):0

Date: / /
Page No.:

* Practical Related Questions:

1] Write the transformation matrix for 2D Translation.

→

$x' = x + t_x$
$y' = y + t_y$

The pair (t_x, t_y) is called translation vector. The above equation can also be represented as:

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

or

$$P' = P + T$$

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

2] Write the transformation for 2D scaling.

→

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

3] What does scaling transformation do?

→ A scaling transformation alters size of an object.

4] Whether size of object remains same or changed in case of translation?

→ The size remains in translation.

* Conclusion:

Transformation are one of the primary vehicles used in Computer graphics used to manipulate 2D and 3D objects.

* Exercise

1] Translate the polygon with co-ordinates A(2, 5), B(7, 10) and C(10, 2) by 3 units in x direction and 4 units in y direction.

→

$$\begin{bmatrix} 2 & 5 & 1 \\ 7 & 10 & 1 \\ 10 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 5 & 1 \\ 7 & 10 & 1 \\ 10 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 4 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 5 & 9 & 1 \\ 10 & 14 & 1 \\ 13 & 6 & 1 \end{bmatrix}$$

A (5, 9)
B (10, 14)
C (13, 6)

2] Scale the polygon with coordinates A(2,5), B(7,10) and C(10,2) by 2 units in x direction and 2 units in y direction.

→

$$\begin{bmatrix} 2 & 5 & 1 \\ 7 & 10 & 1 \\ 10 & 2 & 1 \end{bmatrix} \qquad \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 5 & 1 \\ 7 & 10 & 1 \\ 10 & 2 & 1 \end{bmatrix} \qquad \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 10 & 1 \\ 14 & 20 & 1 \\ 20 & 4 & 1 \end{bmatrix}$$

A(4,10)
B(14,20)
C(20,4)

3] Give a 3×3 homogenious co-ordinate transformation matrix for each of the following transformation.

i] Shift the image to the right 3 units

→

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 1 & 1 \end{bmatrix}$$

2] Shift the image up 2 units.

→

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -2 & 1 \end{bmatrix}$$

3] Move image down 1/2 unit and right 1 unit.

→

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0.5 & 1 \end{bmatrix}$$

4] Move image down 2/3 unit and left 4 unit.

→

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -9 & 0.6 & 1 \end{bmatrix}$$

Date : / /
Page No.:

8. Program for 2 dimensional translation (Rotation)

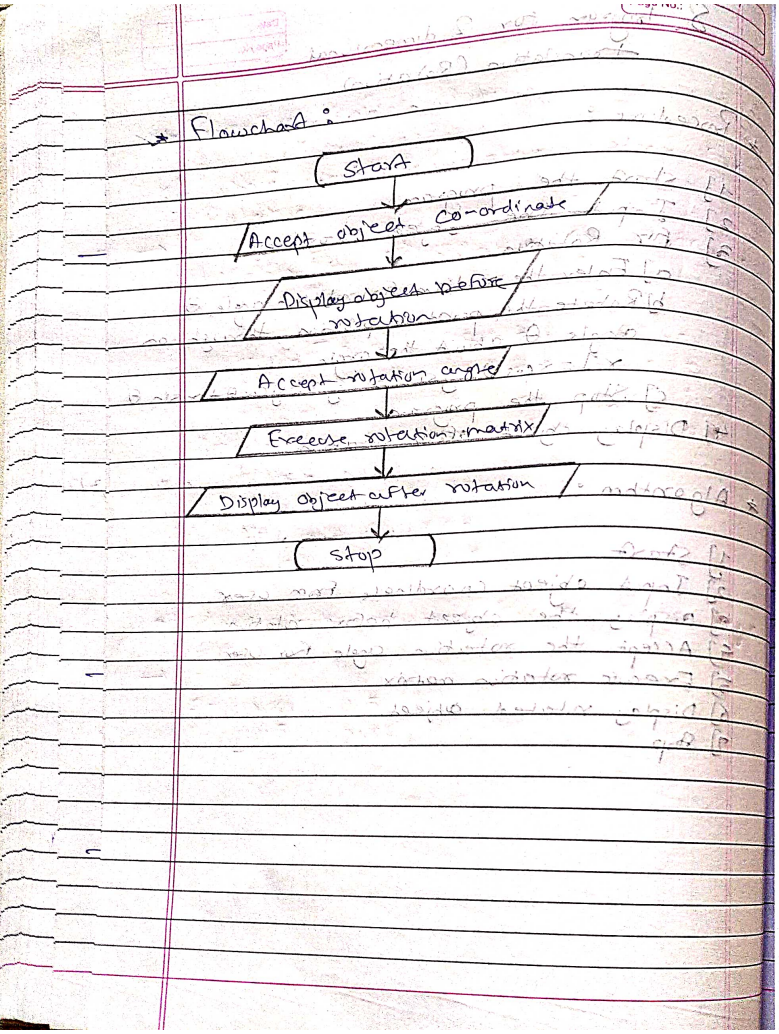
Procedure :

1] Start the program
2] Input the object coordinate
3] For Rotation
   a] Enter the radian for rotation angle θ
   b] Rotate the point at position through an angle θ about the origin
   $x1 = x\cos\theta - y\sin\theta$, $y1 = y\cos\theta + x\sin\theta$
   c] Stop the program.
4] Display object after rotation.

Algorithm :

1] Start
2] Input object coordinate from user.
3] Display the objects before rotation.
4] Accept the rotation agle for user.
5] Execute rotation matrix
6] Display rotated objects
7] Stop

Flowchart :




```
//8 -code
#include<stdio.h>
#include<graphics.h>
#include<math.h>
int main()
{
   intgd=0,gm,x1,y1,x2,y2;
   double s,c, angle;
   initgraph(&gd, &gm, "C:\\TC\\BGI");
   setcolor(RED);
   printf("Enter coordinates of line: ");
   scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
   cleardevice();
   setbkcolor(WHITE);
   line(x1,y1,x2,y2);
   getch();
   setbkcolor(BLACK);
   printf("Enter rotation angle: ");
   scanf("%lf", &angle);
   setbkcolor(WHITE);
   c = cos(angle *3.14/180);
   s = sin(angle *3.14/180);
   x1 = floor(x1 * c + y1 * s);
   y1 = floor(-x1 * s + y1 * c);
   x2 = floor(x2 * c + y2 * s);
   y2 = floor(-x2 * s + y2 * c);
   cleardevice();
   line(x1, y1 ,x2, y2);
   getch();
   closegraph();
return 0;
}
```

Date :
Page No.:

* Practical Related Questions

1] Write the transformation matrix for 2D Rotation.

→

i] Matrix for 2D rotation for clockwise direction.

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

ii] Matrix for 2D rotation for anti-clockwise direction

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

2] Write rotation matrix for a rotation angle 30°.

→

$$R = \begin{bmatrix} 0.86 & 0.5 \\ 0.5 & 0.86 \end{bmatrix}$$

Date :
Page No.:

* Conclusion :

One of the most important task in Computer Graphics is to transform the co-ordinate of object. It is a process by virtue of which the object can rotate to specific angle.

* Exercise.

1] Perform a counterclockwise 45° rotation of triangle A(2,3), B(5,5), C(4,3) about point (1,1).

→

A(2,3) B(5,5) C(4,3)

Angle of rotation = 45°

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0.7 & 0.7 & 0 \\ -0.7 & 0.7 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.7+0+0 & 0.7+0+0 & 0+0+0 \\ 0+0.7+0 & 0+0.7+0 & 0+0+0 \\ 0.7+0+0 & 0.7+0+0 & 0+0+1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.7+0+0 & 0.7+0+0 & 0+0+0 \\ -0.7+0+0 & 0+0.7+0 & 0+0+0 \\ 0.7+0+1 & 0.7+0+1 & 0+0+1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

* Practical Related Questions

1] Write the transformation matrix for 2D Rotation.

→

i] Matrix For 2D rotation For clockwise direction.

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

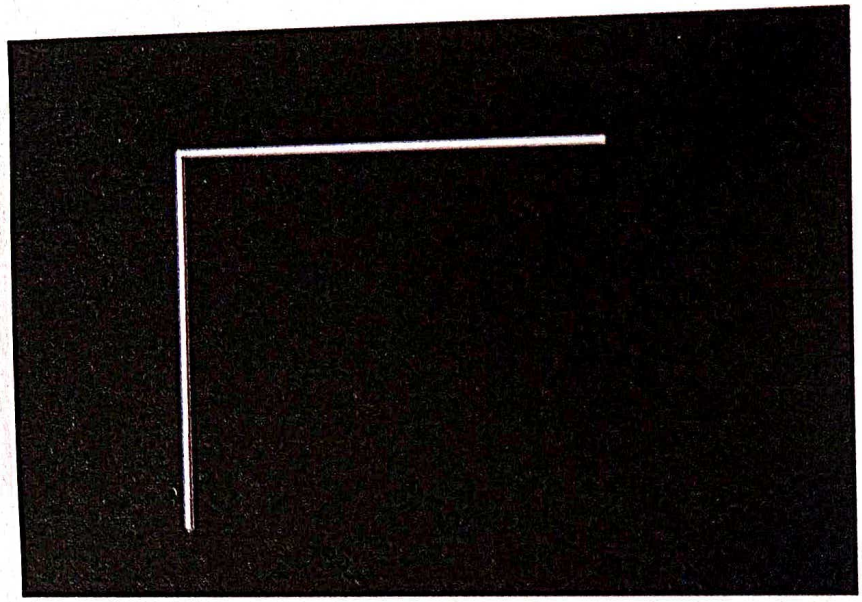
ii] Matrix For 2D rotation For anti-clockwise direction

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

2] Write rotation matrix For a rotation angle 30°.

→

$$R = \begin{bmatrix} 0.86 & 0.5 \\ 0.5 & 0.86 \end{bmatrix}$$



Date : / /
Page No.:

Conclusion :

One of the most important task in Computer Graphics is to transform the co-ordinate of object. It is a process by virtue of which the object can rotate to specific angle.

Exercise

1) Perform a counterclockwise 45° rotation of triangle A(2,3), B(5,5), C(4,3) about point (1,1).

→

A(2,3) B(5,5) C(4,3)

Angle of rotation = 45°

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0.7 & 0.7 & 0 \\ -0.7 & 0.7 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.7+0+0 & 0.7+0+0 & 0+0+0 \\ 0+0.7+0 & 0+0.7+0 & 0+0+0 \\ 0.7+0+0 & 0.7+0+0 & 0+0+1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.7+0+0 & 0.7+0+0 & 0+0+0 \\ -0.7+0+0 & 0+0.7+0 & 0+0+0 \\ 0.7+0+1 & 0.7+0+1 & 0+0+1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.7 & 0.7 & 0 \\ 0.7 & 0.7 & 0 \\ 1.7 & 1.7 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 3 & 1 \\ 5 & 5 & 1 \\ 4 & 3 & 1 \end{bmatrix} * \begin{bmatrix} 0.7 & 0.7 & 0 \\ -0.7 & 0.7 & 0 \\ 1.7 & 1.7 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1.4+2.1+1.7 & 1.4+2.1+1.7 & 0+0+1 \\ 3.5+3.5+1.7 & 3.5+3.5+1.7 & 0+0+1 \\ 2.8+2.1+1.7 & 2.8+2.1+1.7 & 0+0+1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 5.2 & 1 \\ 1.7 & 8.7 & 1 \\ 2.4 & 6.6 & 1 \end{bmatrix}$$

★ Assessment Scheme

9. Program For two dimensional Transformation (Reflection & Shearing)

★ Procedure:

1) Start the program
2) Input the object coordinate
3) For shearing
   a) Input shearing factor
   b) Shearing related to x axis
   c) Shearing related to y axis
   d) Input the xref and yref value
   e) x shear related to reference line
   f) y shear related to reference line
   g) Display object
4) For reflection
   a) Reflection about x-axis
   b) Reflection about y-axis
   c) Display the object after reflection.
5) Stop the Program.

★ Algorithm

1) Shear

1) Input object co-ordinate
2) Input shearing factor for x and y co-ordinate
3) Multiply y-shearing co-ordinate factor with y-coordinate.

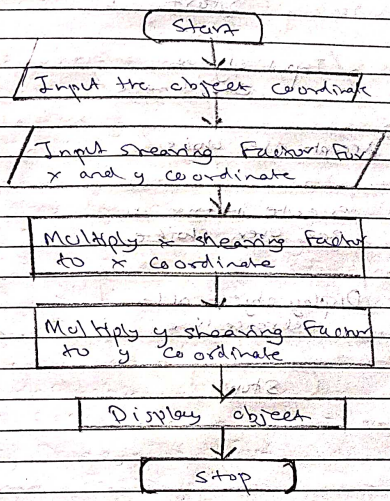
4) display object after shearing
5) Stop

2) Reflection

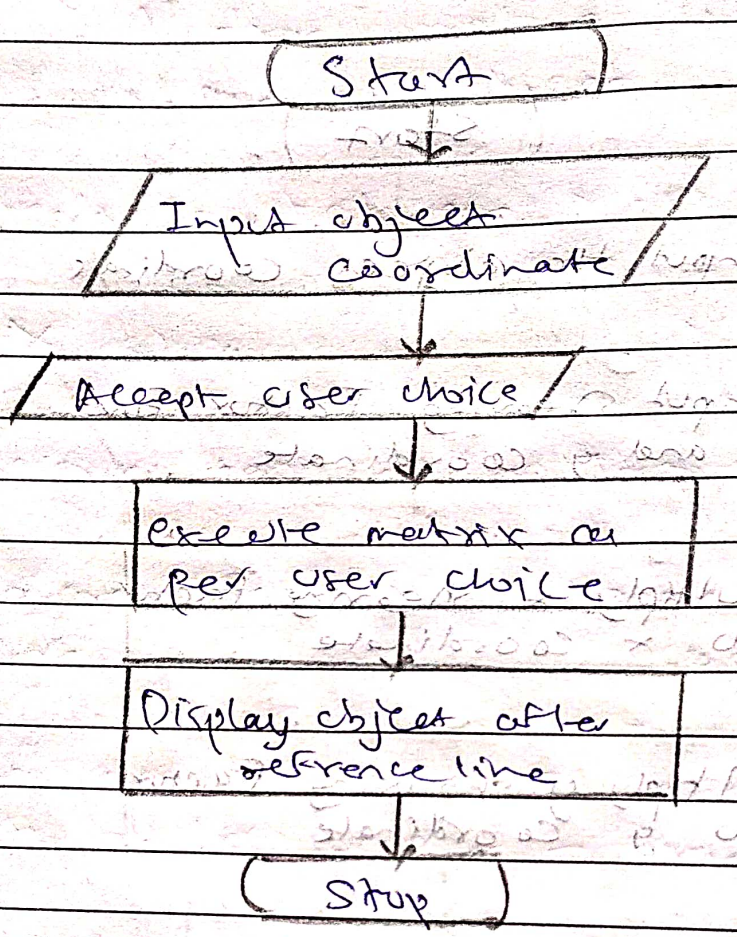
1) Start
2) Input object co-ordinate
3) Ask the choice co-ordinate for user
   a) about x-axis
   b) about y-axis
   c) about origin
   d) x=y
   e) -x=y
4) Perform matrix choose by user
5) display object after reflection
6) Stop

* Flowchart

1) Shear



2) Reflection



```
//9 - reflection
#include <iostream.h>
#include <conio.h>
#include <graphics.h>
#include <math.h>
#include <stdlib.h>
#define pi 3.14
class arc
{
   float x[10],y[10],theta,ref[10][10],ang;
        float p[10][10],p1[10][10],x1[10],y1[10],xm,ym;
   int i,k,j,n;
   public:
   void get();
   void cal ();
   void map ();
   void graph ();
   void plot ();
   void plot1();
};
void arc::get ()
{
   cout<<"\n ENTER ANGLE OF LINE INCLINATION AND Y INTERCEPT";
   cin>> ang >> b;
   cout <<"\n ENTER NO OF VERTICES";
   cin >> n;
   cout <<"\n ENTER";
   for (i=0; i<n; i++)
   {
      cout<<"\n x["<<i<<"] and y["<<i<<"]";
   }
   theta =(ang * pi)/ 180;
   ref [0] [0] = cos (2 * theta);
   ref [0] [1] = sin (2 * theta);
   ref [0] [2] = -b *sin (2 * theta);
   ref [1] [0] = sin (2 * theta);
   ref [1] [1] = -cos (2 * theta);
   ref [1] [2] = b * (cos (2 * theta)+1);
   ref [2] [0]=0;
   ref [2] [1]=0;
   ref [2] [2] = 1;
}
void arc :: cal ()
{
   for (i=0; i < n; i++)
   {
      p[0] [i] = x [i];
```

```
        p [1] [i] = y [i];
        p [2] [i] = 1;
      }
      for (i=0; i<3;i++)
      {
        for (j=0; j<n; j++)
        {
          p1 [i] [j]=0;
          for (k=0;k<3; k++)
        }
        p1 [i] [j] + = ref [i] [k] * p [k] [j];
          }
    for (i=0; i<n; i++)
     {
      x1 [i]=p1[0] [i];
      y1 [i] = p1 [1] [i];
      }
   }
   void arc :: map ()
   {
     int gd = DETECT,gm;
     initgraph (&gd, &gm, " ");
          int errorcode = graphresult ();
     /* an error occurred */
     if (errorcode ! = grOK)
     {
       printf ("Graphics error: %s \n", grapherrormsg (errorcode));
       printf ("Press any key to halt:");
       getch ();
       exit (1); /* terminate with an error code */
     }
  }
  void arc :: graph ()
  {
    xm=getmaxx ()/2;
    ym=getmaxy ()/2;
    line (xm, 0, xmm 2*ym);
  }
  void arc :: plot 1 ()
  {
    for (i=0; i <n-1; i++)
    {
      circle (x1[i]+xm, (-y1[i]+ym), 2);
      line (x1[i]+xm, (-y1[i]+ym), x1[i+1]+xm, (-y1[i+1]+ym));
    }
      line (x1[n-1]+xm, (-y1[n-1]+ym), x1[0]+xm, (-y1[0]+ym));
      getch();
```

```
        p [1] [i] = y [i];
        p [2] [i] = 1;
    }
    for (i=0; i<3;i++)
    {
       for (j=0; j<n; j++)
       {
          p1 [i] [j]=0;
          for (k=0;k<3; k++)
       }
       p1 [i] [j] + = ref [i] [k] * p [k] [j];
           }
for (i=0; i<n; i++)
  {
   x1 [i]=p1[0] [i];
   y1 [i] = p1 [1] [i];
   }
}
void arc :: map ()
{
   int gd = DETECT,gm;
   initgraph (&gd, &gm, " ");
         int errorcode = graphresult ();
   /* an error occurred */
   if (errorcode ! = grOK)
   {
      printf ("Graphics error: %s \n", grapherrormsg (errorcode));
      printf ("Press any key to halt:");
      getch ();
      exit (1); /* terminate with an error code */
   }
}
void arc :: graph ()
{
  xm=getmaxx ()/2;
  ym=getmaxy ()/2;
  line (xm, 0, xmm 2*ym);
}
void arc :: plot 1 ()
{
  for (i=0; i <n-1; i++)
  {
     circle (x1[i]+xm, (-y1[i]+ym), 2);
     line (x1[i]+xm, (-y1[i]+ym), x1[i+1]+xm, (-y1[i+1]+ym));
  }
     line (x1[n-1)+xm, (-y1[n-1]+ym), x1[0]+xm, (-y1[0]+ym));
     getch();
```

```
}
void arc :: plot ()
{
   for (i=0; i <n-1; i++)
   {
      circle (x1[i]+xm, (-y1[i]+ym, 2);
      line (x1[i]+xm, (-y1[i]+ym), x[i+1]+xm, (-y1[i+1]+ym));
   }
      line (x[n-1]+xm, (-y1[n-1]+ym), x[0]+xm, (-y[0]+ym));
      getch();
}
void main ()
{
   class arc a;
   clrscr();
   a.map();
   a.graph();
   a.get();
   a.cal();
   a.plot();
   a.plot1();
   getch();
}
```

Date : / /

```
//9 - shearing
#include<iostream.h>
#include<graphics.h>
#include<math.h>
#include<conio.h>
#include<dos.h>
void mul(int mat[3][3],int vertex[10][3],int n);
void shear(int vertex[10][3],int n);
void init(int vertex[10][3],int n);
int main()
{
 int i,x,y;
 int vertex[10][3],n;
 clrscr();
 cout<<"\nEnter the no. of vertex : ";
 cin>>n;
  for(i=0;i<n;i++)
   {
            cout<<"Enter the points (x,y): ";
            cin>>x>>y;
            vertex[i][0]=x;
            vertex[i][1]=y;
            vertex[i][2]=1;
   }
   shear(vertex,n);

          getch();
          return 0;
}
void init(int vertex[10][3],int n)
{
  int gd=DETECT,gm,i;
  initgraph(&gd,&gm,"C:\\turboc3\\bgi");
  setcolor(10);
  line(0,240,640,240);
  line(320,0,320,480);
  setcolor(3);
  line(450,20,490,20);
  setcolor(15);
  line(450,50,490,50);
  setcolor(6);
  outtextxy(500,20,"Original");
  outtextxy(500,50,"Transformed");
  setcolor(3);

  for(i=0;i<n-1;i++)
  {
```

```
    line(320+vertex[i][0],240-vertex[i][1],320+vertex[i+1][0],240-vertex[i+1][1]);
   }
   line(320+vertex[n-1][0],240-vertex[n-1][1],320+vertex[0][0],240-vertex[0][1]);

}
void mul(int mat[3][3],int vertex[10][3],int n)
{
 int i,j,k;
 int res[10][3];
 for(i=0;i<n;i++)
 {
   for(j=0;j<3;j++)
    {
                res[i][j]=0;
      for(k=0;k<3;k++)
                {
                  res[i][j] = res[i][j] + vertex[i][k]*mat[k][j];
                }
    }
   }
   setcolor(15);
   for(i=0;i<n-1;i++)
   {
     line(320+res[i][0],240-res[i][1],320+res[i+1][0],240-res[i+1][1]);
    }
    line(320+res[n-1][0],240-res[n-1][1],320+res[0][0],240-res[0][1]);

 }
 void shear(int vertex[10][3],int n)
 {
   int opt;
   int shear_array[3][3];
  cout<<"\n1.x-shear\n2.y-shear\nYour Choice: ";
   cin>>opt;
   switch(opt)
   {
    case 1: int xsh;
                        cout<<"\nEnter the x shear : ";
                cin>>xsh;
                        shear_array[0][0]=1;
              shear_array[1][0]=xsh;
              shear_array[2][0]=0;
              shear_array[0][1]=0;
              shear_array[1][1]=1;
              shear_array[2][1]=0;
              shear_array[0][2]=0;
              shear_array[1][2]=0;
```

```
              shear_array[2][2]=1;
              init(vertex,n);
              mul(shear_array,vertex,n);
               break;

            case 2:int ysh;
                 cout<<"\nEnter the y shear : ";
                 cin>>ysh;
                        shear_array[0][0]=1;
              shear_array[1][0]=0;
              shear_array[2][0]=0;
              shear_array[0][1]=ysh;
               shear_array[1][1]=1;
               shear_array[2][1]=0;
               shear_array[0][2]=0;
               shear_array[1][2]=0;
               shear_array[2][2]=1;
                init(vertex,n);
                 mul(shear_array,vertex,n);
                  break;
    }
}
```

Date : / /
Page No.:

* Practical Related Questions :

1] Write the transformation matrix for 2D Reflection.

→

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

2] Write the transformation matrix for 2D shear

→ x Shear :

$$X_{sh} = \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Y Shear :

$$Y_{sh} = \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3] Define reflection and shearing.

→

Reflection :

It is the change in direction of a wave at the boundary between two different media, so the wave moves back to its medium.

Shearing:
A transformation which changes the shape of object.

4] Difference between x-shear and Y shear

X shear	Y shear
1] It preserves the Y coordinate	It preserves the x coordinate
2] Changes are made to x coordinate	Changes are made to Y coordinate
3] Change cause the vertical line to tilt left or right	It causes horizontal line to transform up or down

★ Conclusion:

Thus, we learned the program of reflection and shearing of two dimensional object

★ Exercise

1] A point (4,3) is rotated counterclockwise by an angle of 45°. Find the rotation matrix and the resultant point. Apply the Shearing transformation to square with A(0,0), B(1,0) C(1,1) D(0,1) as below:

→

1] Shear parameter value of 0.5 relative to line $y_{ref} = -1$

→

1] A(0,0), B(1,0) C(1,1) D(0,1)
shear parameter = 0.5
and $y_{ref} = -1$

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ Shx & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 1 \\ 2 & 0 & 1 \\ 2.5 & 1 & 1 \\ 1.5 & 1 & 1 \end{bmatrix}$$

10. Program For three Dimensional Transformation (Translation & Scaling)

* Procedure:

1) Start the Program
2) Input object Coordinate
3) For translation
   i) Enter translation factor t_x, t_y and t_z
   ii) Move original position to new position
   iii) Display the object after translation
4) For Scaling
   i) Input the scaled factors S_x, S_y, and S_z
   ii) The transformed co-ordinate
   iii) Display the objects
5) Stop the program.

* Algorithm

1) Translation

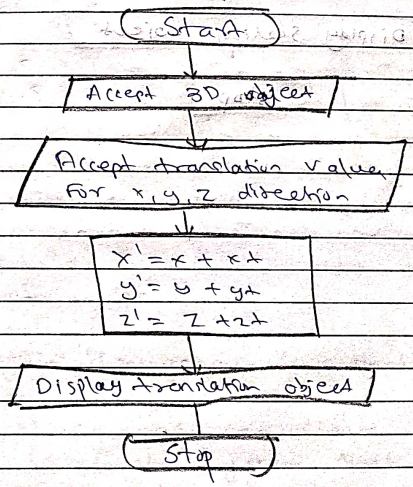
i) Start
ii) Accept object coordinate
iii) Accept translation values
iv) Add translation values to original coordinate
v) Display object after translation.
vi) Stop

2) Scaling

i) Start
ii) Accept object co-ordinate
iii) Accept Scaling values
iv) Multiply Scaling values to original coordinate
v) Display Scaled objects
vi) Stop

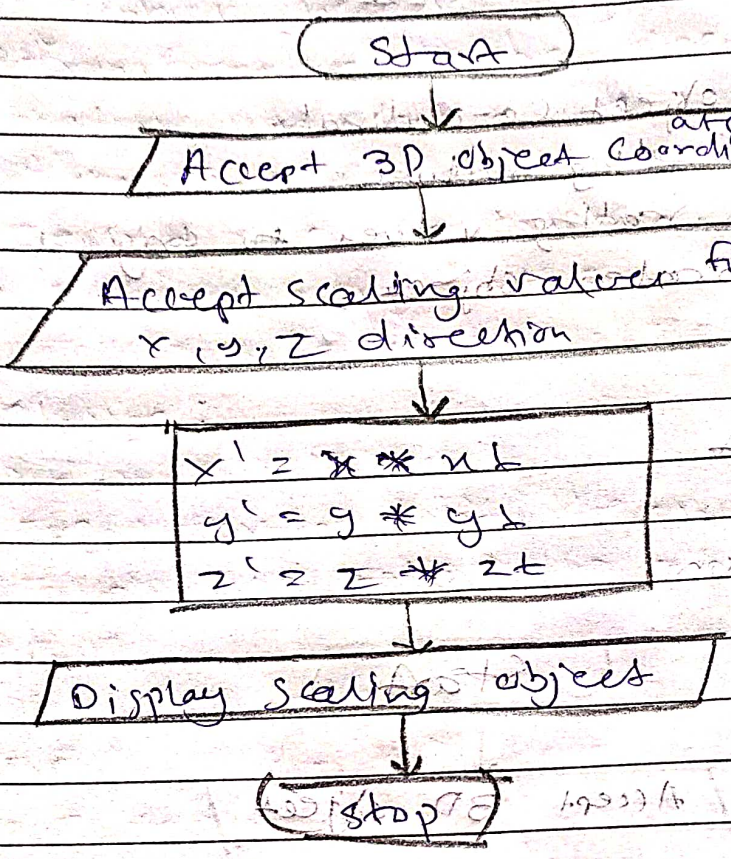
* Flowchart

1) Translation

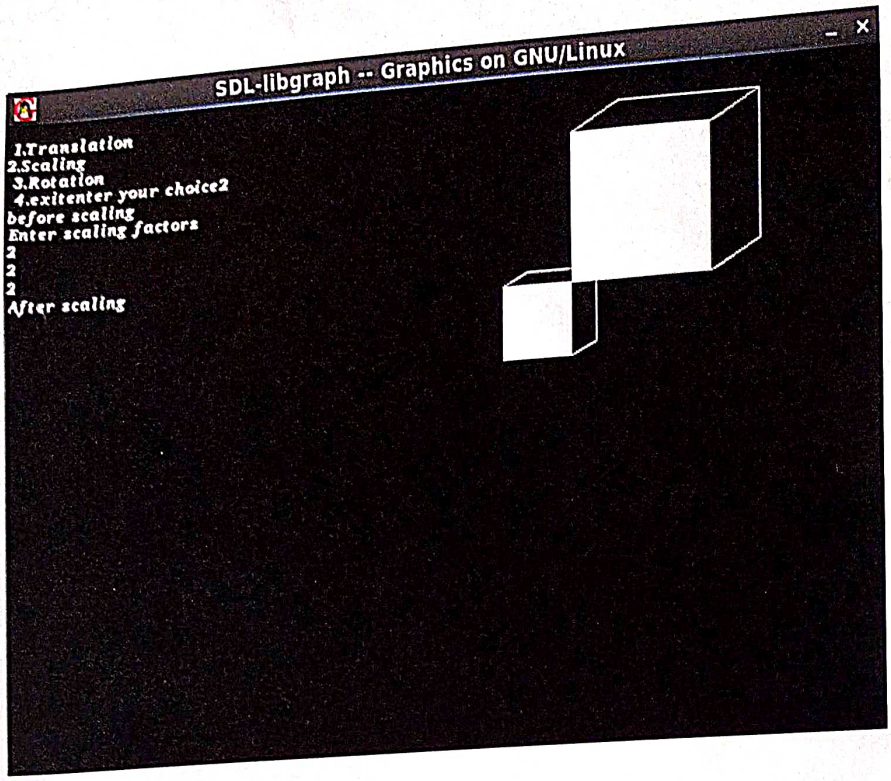


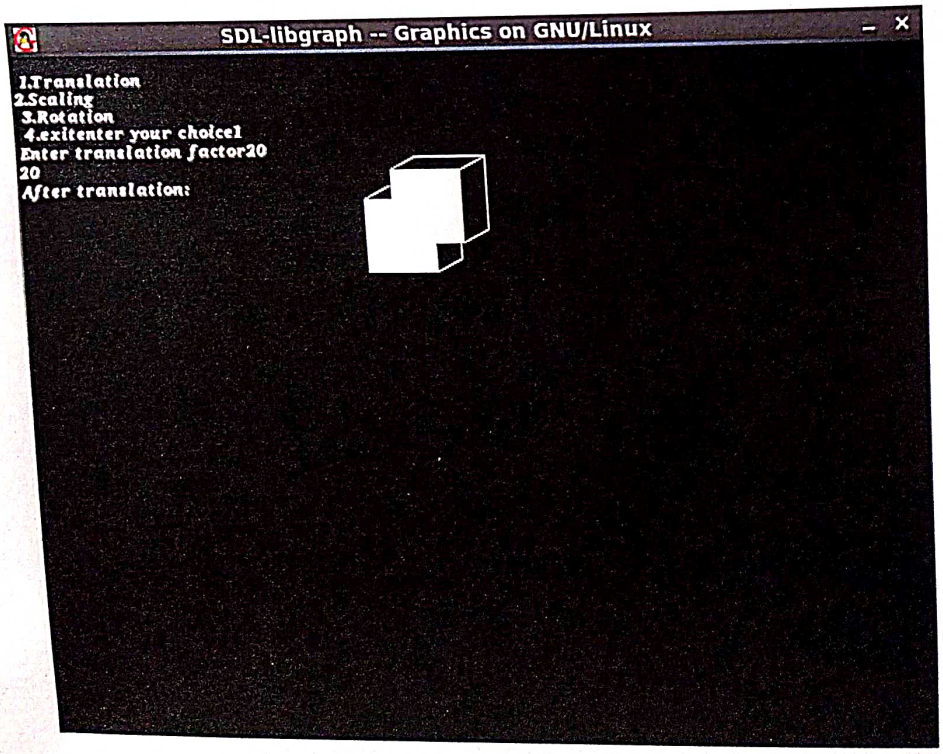
Date : / /
Page No.:

2] Scaling



```
// 10 - translation
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
int maxx,maxy,midx,midy;
void axis()
{
getch();
cleardevice();
line(midx,0,midx,maxy);
line(0,midy,maxx,midy);
}
void main()
{
int x,y,z,o,x1,x2,y1,y2;
int gd=DETECT,gm;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"c:\\tc\\bgi");
//setfillstyle(0,getmaxcolor());
maxx=getmaxx();
maxy=getmaxy();
midx=maxx/2;
midy=maxy/2;

axis();

bar3d(midx+50,midy-100,midx+60,midy-90,10,1);

printf("Enter translation factor");
scanf("%d%d",&x,&y);
//axis();
printf("After translation:");
bar3d(midx+x+50,midy-(y+100),midx+x+60,midy-(y+90),10,1);
getch();
closegraph();
}
```


SDL-libgraph -- Graphics on GNU/Linux
1.Translation
2.Scaling
3.Rotation
4.exitenter your choice2
before scaling
Enter scaling factors
2
2
2
After scaling


SDL-libgraph -- Graphics on GNU/Linux
1.Translation
2.Scaling
3.Rotation
4.exitenter your choice1
Enter translation factor20
20
After translation:

Date : / /
Page No.:

★ Practical Related Questions :

1] Write the transformation matrix for 3D Translation.

→

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

2] Write transformation matrix for 3D Scaling.

→

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3] What Z-coordinate indicates in 3D transformation.

→ Z-coordinate indicates rotation about front to back rotation in 3D object transformation.

4] Explain Homogenous Coordinate

→

i] Homogeneous coordinate is a coordinate system that algebraically treats all points in the projective plane equally.

ii) They are widely used in computer graphics because they enable physical transformation to be described in matrix manipulation form in a coherent way.

iii) Homogeneous coordinate system is used in constructions and design based applications.

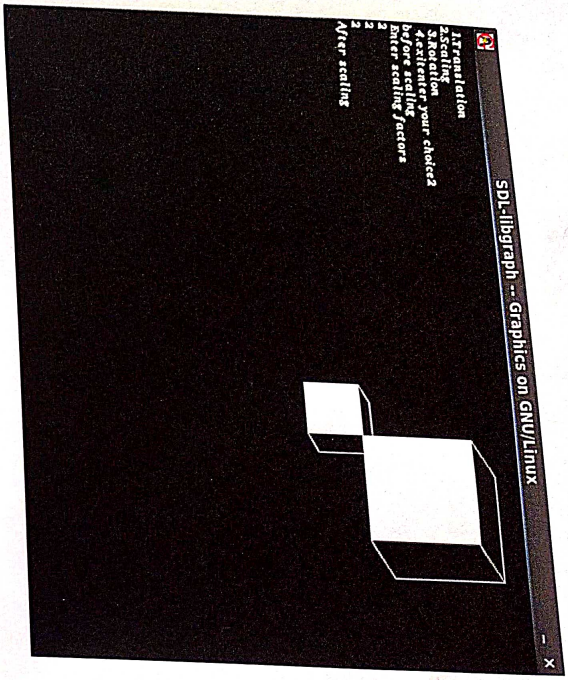
* Conclusion:

Thus, we learned that the program to translate and scale the three dimensional object.

* Exercise

1) Write a program to draw a cube and cuboid by using bar3d function. Translate the cube by 75 units in X, 75 units in Y and 75 units in Z directions.

10-2



11. Program For three dimentional Transformation (Rotation)

Date : / /
Page No.:

Procedure :

1] Start the program
2] Input the object co-ordinate
3] For rotation
a] Enter the Radian for rotation angle θ
b] Perform rotation about each axis
i] Z-Axis Rotation

$x' = x * \cos\theta - y * \sin\theta$
$y' = x * \sin\theta + y * \cos\theta$
$z' = z$

ii] X-Axis Rotation

$y' = y * \cos\theta - z * \sin\theta$
$z' = y * \sin\theta + z * \cos\theta$
$x' = x$

iii] Y-Axis Rotation

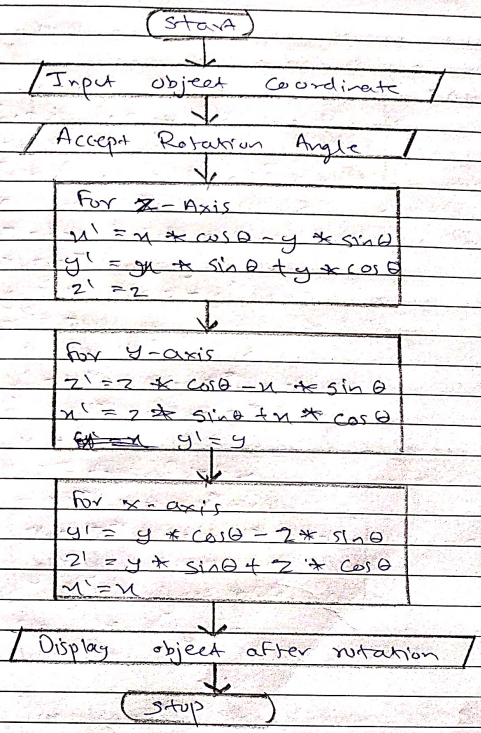
$z' = z * \cos\theta - x * \sin\theta$
$x' = z * \sin\theta + x * \cos\theta$
$y' = y$

c] Display the object after rotation.
4] Stop the program

★ Algorithm :

1] Start
2] Input object coordinate.
3] Accept rotation angle.
4] For z-axis rotation
$x' = x * \cos\theta - y * \sin\theta$
$y' = x * \sin\theta + y * \cos\theta$
$z' = z$
5] For x-axis rotation
$y' = y * \cos\theta - z * \sin\theta$
$z' = y * \sin\theta + z * \cos\theta$
$x' = x$
6] For y-axis rotation
$z' = z * \cos\theta - x * \sin\theta$
$x' = z * \sin\theta + x * \cos\theta$
$y' = y$
7] Display object after rotation
8] Stop

★ Flowchart :



```
// 11 - rotation
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
int maxx,maxy,midx,midy;
void axis()
{
getch();
cleardevice();
line(midx,0,midx,maxy);
line(0,midy,maxx,midy);
}
void main()
{
int x,y,z,o,x1,x2,y1,y2;
int gd=DETECT,gm;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"c:\\tc\\bgi");
//setfillstyle(0,getmaxcolor());
maxx=getmaxx();
maxy=getmaxy();
midx=maxx/2;
midy=maxy/2;
axis();
bar3d(midx+50,midy-100,midx+60,midy-90,5,1);
printf("Enter rotating angle");
scanf("%d",&o);
 x1=50*cos(o*3.14/180)-100*sin(o*3.14/180);
 y1=50*sin(o*3.14/180)+100*cos(o*3.14/180);
 x2=60*cos(o*3.14/180)-90*sin(o*3.14/180);
 y2=60*sin(o*3.14/180)+90*cos(o*3.14/180);
 axis();
printf("After rotation about z axis");
bar3d(midx+x1,midy-y1,midx+x2,midy-y2,5,1);
axis();
printf("After rotation  about x axis");
bar3d(midx+50,midy-x1,midx+60,midy-x2,5,1);
axis();
printf("After rotation about yaxis");
bar3d(midx+x1,midy-100,midx+x2,midy-90,5,1);
getch();
closegraph();
}
```

Date : / /
Page No.:

* Practical Related Questions

1] Write the transformation matrix for 3D Rotation about Z - axis.

→

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

2] Write the transformation matrix for 3D Rotation about x - axis.

→

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

3] Write the transformation matrix for 3D Rotation about Y - axis

→

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

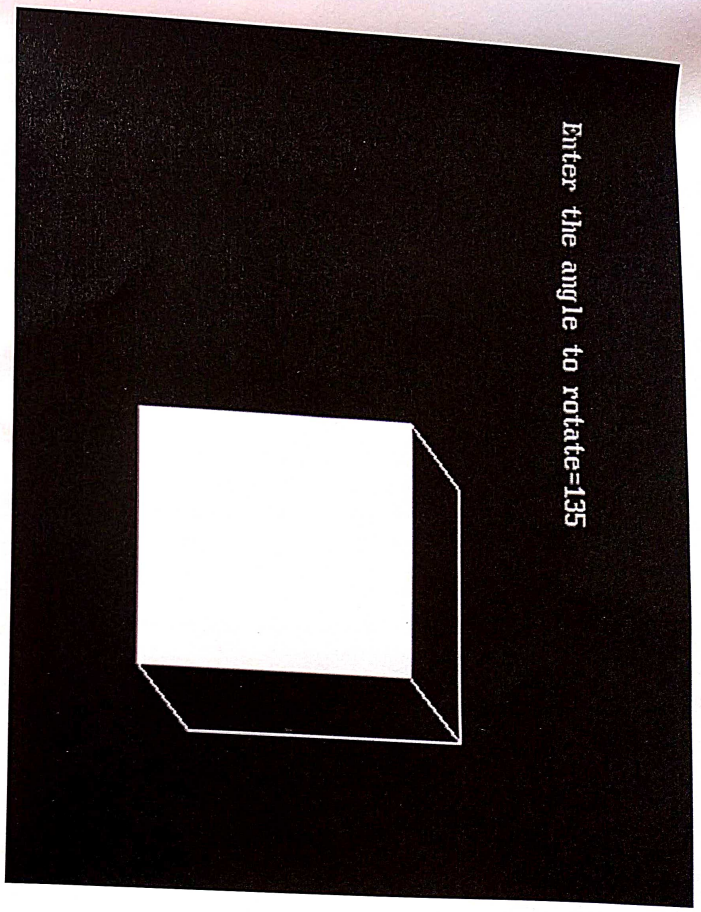
* Conclusion :

Thus, we learned the program to rotate the 3 dimensional object.

Date :
Page No.:

* Exercise:

1) WAP to draw a cube in C by using 'bar3d' function. Rotate the cube by 45° around x-axis.



Date : / /
Page No.:

12. Program to Clip line using Cohen Sutherland line Clipping Algorithm.

* Procedure :

1] Given a line segment with endpoint $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$
2] Compute the 4 bit code.
3] If both code are 0000, line lies completely inside the window.
4] If both code have 1 in same bit position, the line lies outside the border.
5] If a line cannot be trivially be accepted or rejected, at least one of the two endpoint must lie outside the window and the line segment crosses a window edge. This line must be clipped at the window edge before being passed to the drawing machine.
6] Examine one of the endpoint, say $P_1(x_1, y_1)$. Read P_1's 4 bit code.
7] When a set bit is found, compute the intersection I of the corresponding window edge with the line from P_1 to P_2. Replace P_1 with I and repeat the procedure.

Date : / /
Page No.:

Algorithm

1] Calculate positions of both endpoints of the line

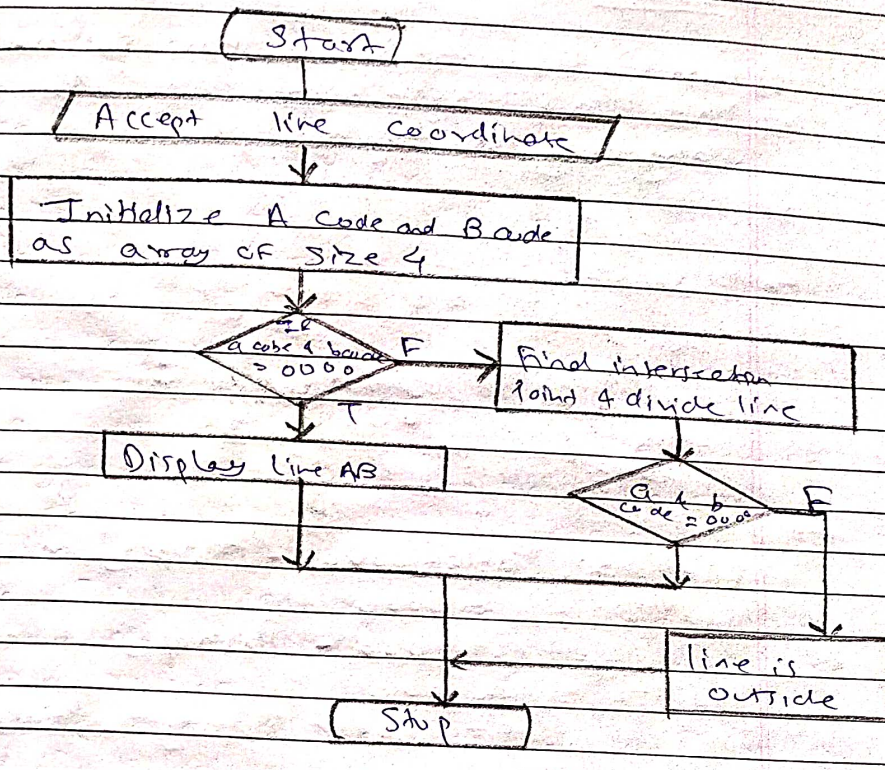
2] Perform OR operation on both of these endpoint

3] IF OR give 0000
then
line is visible
else
Perform AND operation
IF AND ≠ 0000
then
line is invisible
else
And = 0000
Line is considered clipped.

4] Stop

Date : / /
Page No.:

Flowchart



```
// 12 -code
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<graphics.h>
#include<dos.h>

typedef struct coordinate
{
        int x,y;
        char code[4];
}PT;

void drawwindow();
void drawline(PT p1,PT p2);
PT setcode(PT p);
int visibility(PT p1,PT p2);
PT resetendpt(PT p1,PT p2);

void main()
{
        int gd=DETECT,v,gm;
        PT p1,p2,p3,p4,ptemp;
        printf("\nEnter x1 and y1\n");
        scanf("%d %d",&p1.x,&p1.y);
        printf("\nEnter x2 and y2\n");
        scanf("%d %d",&p2.x,&p2.y);
        initgraph(&gd,&gm,"c:\\turboc3\\bgi");
        drawwindow();
        delay(500);
        drawline(p1,p2);
        delay(500);
        cleardevice();
        delay(500);
        p1=setcode(p1);
        p2=setcode(p2);
        v=visibility(p1,p2);
        delay(500);
        switch(v)
        {
        case 0: drawwindow();
                        delay(500);
                        drawline(p1,p2);
                        break;
        case 1:  drawwindow();
                        delay(500);
                        break;
```

```
            case 2:    p3=resetendpt(p1,p2);
                       p4=resetendpt(p2,p1);
                       drawwindow();
                       delay(500);
                       drawline(p3,p4);
                       break;
            }
            delay(5000);
            closegraph();
}
 void drawwindow()
{
            line(150,100,450,100);
            line(450,100,450,350);
            line(450,350,150,350);
            line(150,350,150,100);
}
 void drawline(PT p1,PT p2)
{
            line(p1.x,p1.y,p2.x,p2.y);
}
 PT setcode(PT p)
{
            PT ptemp;

            if(p.y<100)
                        ptemp.code[0]='1'; //Top
            else
                        ptemp.code[0]='0';

            if(p.y>350)
                        ptemp.code[1]='1'; //Bottom
            else
                        ptemp.code[1]='0';

            if(p.x>450)
                        ptemp.code[2]='1'; //Right
            else
                        ptemp.code[2]='0';

            if(p.x<150)
                        ptemp.code[3]='1'; //Left
            else
                        ptemp.code[3]='0';

           ptemp.x=p.x;
           ptemp.y=p.y;

           return(ptemp);
}
 int visibility(PT p1,PT p2)
{
            int i,flag=0;

            for(i=0;i<4;i++)
            {
                        if((p1.code[i]!='0') || (p2.code[i]!='0'))
                                    flag=1;
            }

            if(flag==0)
                        return(0);

            for(i=0;i<4;i++)
            {
                        if((p1.code[i]==p2.code[i]) && (p1.code[i]=='1'))
                                    flag='0';
            }

            if(flag==0)
                        return(1);

            return(2);
}
 PT resetendpt(PT p1,PT p2)
{
            PT temp;
            int x,y,i;
            float m,k;

            if(p1.code[3]=='1')
                        x=150;

            if(p1.code[2]=='1')
                        x=450;

            if((p1.code[3]=='1') || (p1.code[2]=='1'))
            {
                        m=(float)(p2.y-p1.y)/(p2.x-p1.x);
                        k=(p1.y+(m*(x-p1.x)));
                        temp.y=k;
                        temp.x=x;
```

```
                for(i=0;i<4;i++)
                        temp.code[i]=p1.code[i];

                if(temp.y<=350 && temp.y>=100)
                        return (temp);
        }

        if(p1.code[0]=='1')
                y=100;

        if(p1.code[1]=='1')
                y=350;

        if((p1.code[0]=='1') || (p1.code[1]=='1'))
        {
                m=(float)(p2.y-p1.y)/(p2.x-p1.x);
                k=(float)p1.x+(float)(y-p1.y)/m;
                temp.x=k;
                temp.y=y;

                for(i=0;i<4;i++)
                        temp.code[i]=p1.code[i];

                return(temp);
        }
        else
                return(p1);
}
```

Date : / /
Page No.:

★ Practical Related Questions.

i] Define Clipping.

→ Clipping is a method of discarding the unwanted part of image or object from view port.

ii] How to calculate intersection points of a line if line is partially visible

→ Replace endpoint with the intersection point and update the region code.

iii] Give 3 possible condition to clip line using algorithm.

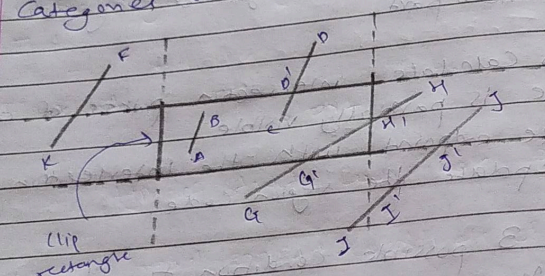
→ i] Completely visible
ii] Completely invisible
iii] Partially visible

★ Conclusion :

Thus, we learned the program to clip line using Cohen Sutherland line Clipping algorithm.

* Exercise:

1) Write four bit code for following lines and determine the clipping categories of each line.



Line AB: A=B=0000 - Totally inside

line CD: C = 0000
D = 1000 - Partially visible

line EF: E = 0001
F = 1001 - Totally outside

line GH: G = 0100
H = 0000 - Partially Inside
Partially outside

Line IJ: I = 0100
J = 0010 - totally outside

13. Program to Clip line

Using midpoint subdivision line Clipping Algorithm

* Procedure

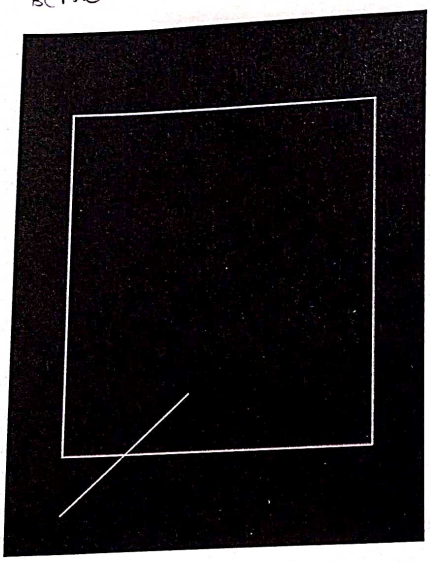
1) Give a line segment with endpoint
2) Four digit for points P1 and P2 are 1000 & 0010 resp.
3) Anding of four digit codes is 0000 line P1P2 is partially visible
4) Divide line P1P2 at midpoint P3 forming two segment P1P3 and P3P2, apply visibility test on both segment.
5) Both lines are partially visible category, so we have to divide it again.
6) This procedure continues until all the visible and invisible line are formed.

* Algorithm

1) Start
2) Read two endpoint of line.
3) Assign region code using two endpoint by following steps:
   initialize code with bit 0000
   set bit 1: if (x < wx1)
   set bit 2: if (x < wx2)
   set bit 3: if (y < wy1)
   set bit 4: if (y < wy2)

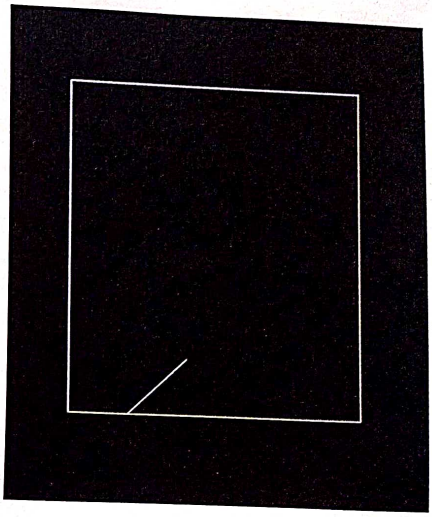
- 1

Before clipping



in-2

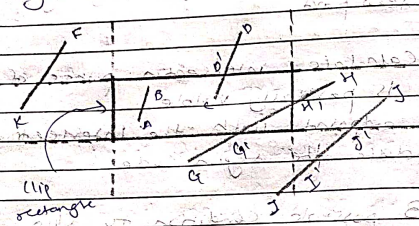
After clipping



Date: / /
Page No.:

* Exercise:

1) Write four bit code for following lines and determine line clipping categories of each line.



→

Line AB: A=B=0000 — Totally inside.
line CD: C = 0000
D = 1000 — Partially visible
line EF: E = 0001
F = 1001 — Totally outside
line GH: G = 0100
H = 0000 — Partially inside
Partially outside
Line IJ: I = 0100
J = 0010 — totally outside

Date: / /
Page No.:

13. Program to Clip line

Using midpoint subdivision line Clipping Algorithm

* Procedure

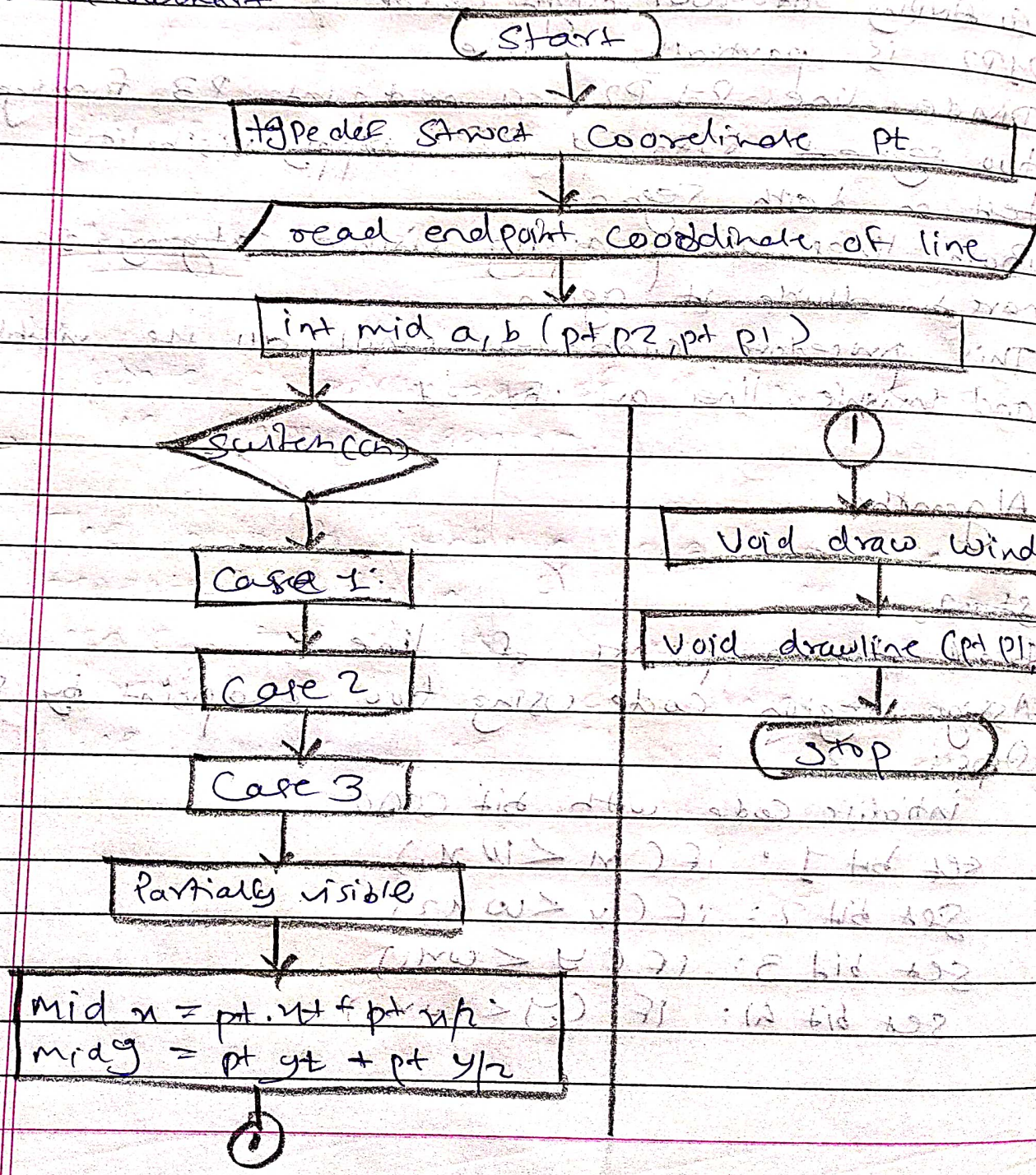
1) Give a line segment with endpoints
2) Four digit for points P1 and P2 are 1000, 0010 resp.
3) On ANDing of four digit codes is 0000 line P1P2 is partially visible
4) Divide line P1 P2 at midpoint P3 forming two segment P1P3 and P3P2, apply visibility test on both segment
5) Both lines are partially visible category, so we have to divide it again
6) This procedure continues until all the visible and invisible line are formed.

* Algorithm

1) Start
2) Read two endpoints of line.
3) Assign region code using two endpoint by following steps:
   initialize code with bit 0000
   Set bit 1: if (x < wx1)
   Set bit 2: if (x < wx2)
   Set bit 3: if (y < wy1)
   Set bit 4: if (y < wy2)

4) Check for visibility of line.
   a) if region code for both endpoints are zero
   b) if region code for both endpoints are not zero
5) Divide Partial line with you get a completely visible segment by using 3rd step.
6) End.

* Flowchart



```
//13 - code
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<math.h>
#include<graphics.h>
typedef struct coordinate
{
int x,y;
char code[4];
}PT;
void drawwindow();
void drawline (PT p1,PT p2,int cl);
PT setcode(PT p);

int visibility (PT p1,PT p2);
PT resetendpt (PT p1,PT p2);
main()
{
 int gd=DETECT, gm,v;
 PT p1,p2,ptemp;
 initgraph(&gd,&gm,"C:\\TC\\BGI ");
 cleardevice();
 printf("\n\n\t\tENTER END-POINT 1 (x,y): ");
 scanf("%d,%d",&p1.x,&p1.y);
 printf("\n\n\t\tENTER END-POINT 2 (x,y): ");
 scanf("%d,%d",&p2.x,&p2.y);
 cleardevice();
 drawwindow();
 getch();
 drawline(p1,p2,15);
 getch();
 cleardevice();
 drawwindow();
 midsub(p1,p2);
 getch();
 closegraph();
 return(0);
}

midsub(PT p1,PT p2)
{
 PT mid;
 int v;
 p1=setcode(p1);
 p2=setcode(p2);
```

```
v=visibility(p1,p2);
switch(v)
 {
 case 0:  /* Line conpletely visible */
   drawline(p1,p2,15);
   break;
 case 1:  /* Line completely invisible */
   break;
 case 2:  /* line partly visible */
   mid.x = p1.x + (p2.x-p1.x)/2;
   mid.y = p1.y + (p2.y-p1.y)/2;
   midsub(p1,mid);
   mid.x = mid.x+1;
   mid.y = mid.y+1;
   midsub(mid,p2);
   break;
  }
}

void drawwindow()
{
setcolor(RED);
line(150,100,450,100);
line(450,100,450,400);
line(450,400,150,400);
line(150,400,150,100);
}

void drawline (PT p1,PT p2,int cl)
{
setcolor(cl);
line(p1.x,p1.y,p2.x,p2.y);
}

PT setcode(PT p)
{
PT ptemp;
if(p.y<=100)
ptemp.code[0]='1'; /* TOP */
else
ptemp.code[0]='0';
if(p.y>=400)
ptemp.code[1]='1'; /* BOTTOM */
else
ptemp.code[1]='0';
if (p.x>=450)
ptemp.code[2]='1'; /* RIGHT */
else
ptemp.code[2]='0';
if (p.x<=150)     /* LEFT */
ptemp.code[3]='1';
else
ptemp.code[3]='0';
ptemp.x=p.x;
ptemp.y=p.y;
return(ptemp);
}

int visibility (PT p1,PT p2)
{
int i,flag=0;
for(i=0;i<4;i++)
{
if((p1.code[i]!='0')||(p2.code[i]!='0'))
flag=1;
}
if(flag==0)
return(0);
for(i=0;i<4;i++)
{
if((p1.code[i]==p2.code[i]) &&(p1.code[i]=='1'))
flag=0;
}
if(flag==0)
return(1);
return(2);
}
```

Date : / /
Page No.:

Practical Related Questions

1) Calculate coordinate of midpoint between two points $P_1(-10, 20)$ and $P_2(50, 10)$.

→

$(x_1, y_1) = (-10, 20)$

$(x_2, y_2) = (50, 10)$

$(x_{min}, y_{min}) = (100, 400)$

$(x_{max}, y_{max}) = (400, 400)$

$P_1 = \Delta x = x_2 - x_1 = (50 - (-10)) = 60$

$P_2 = 60$

$P_3 = 10$

$P_4 = -10$

$q_1 = 410$

$q_2 = 410$

$q_3 = -80$

$q_4 = 380$

$\frac{q_1}{p_1} = \frac{410}{60} = 6.8333$

$\frac{q_2}{p_2} = \frac{410}{60} = 6.8333$

$\frac{q_3}{p_3} = \frac{-80}{10} = -8$

$\frac{q_4}{p_4} = \frac{380}{-10} = -38$

$p_1 = 0 \Rightarrow t = \max(-38, -8)$

$p_2 = 0 \Rightarrow t = \min(8.333, -1.8333)$

$x_1 = x_1 + t_1 \Delta x$

$= -10 + 6 \rightarrow 8.333 * 60$

$= 39.9$

$y = 92.9$

$x_2 = -90 \quad y_2 = -60$

2] Write disadvantage of Midpoint subdivision line clipping algorithm

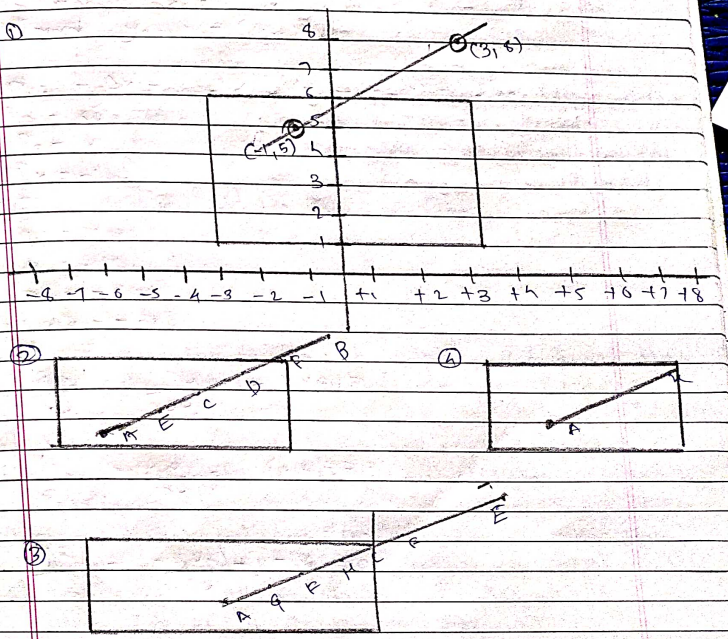
→

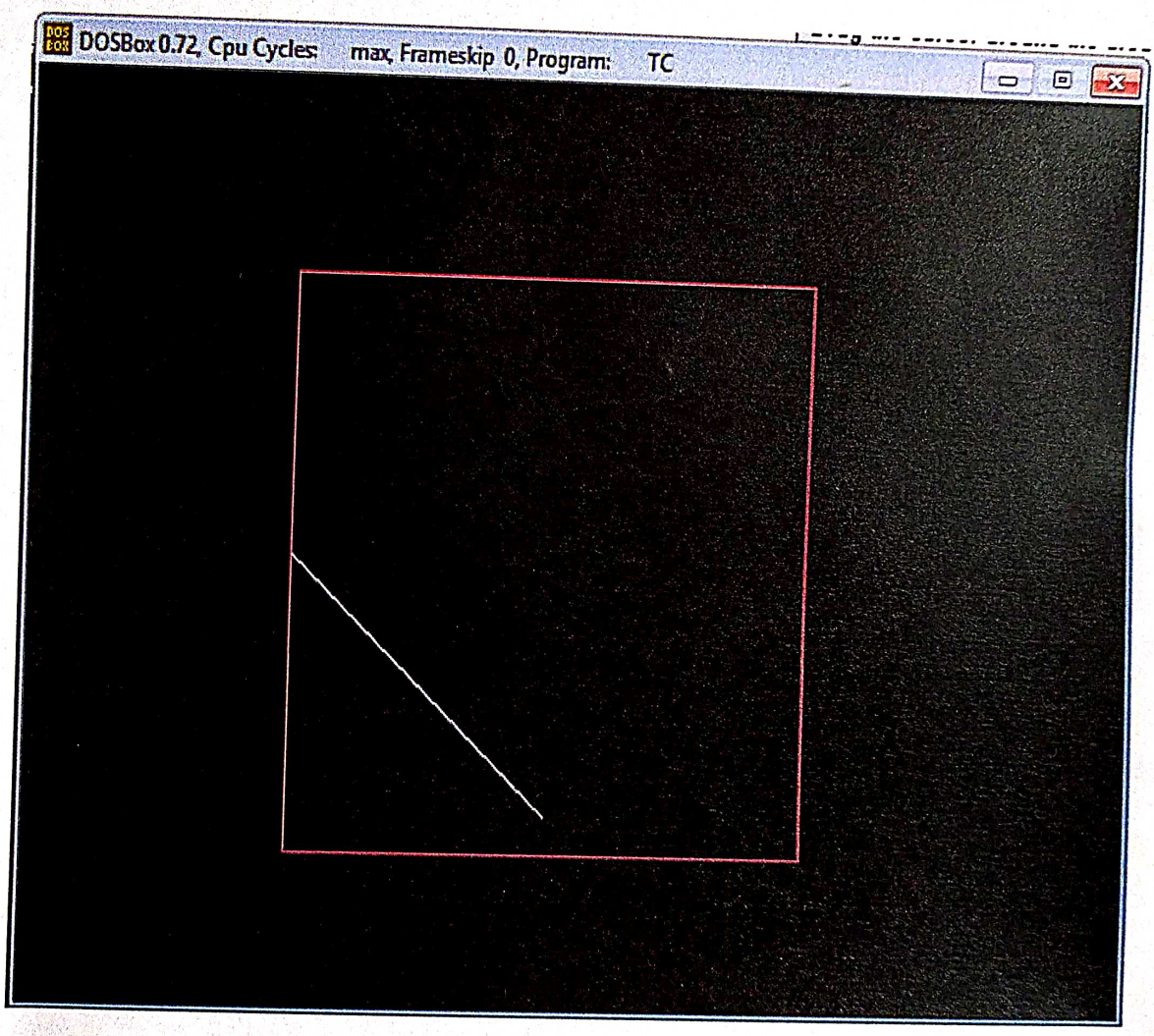
The midpoint subdivision algorithm requires repeated subdivision of the line segment and hence it is slower than using direct calculation of intersection of line with the clipping window edges.

→ Exercise :

1] Using midpoint subdivision line clipping algorithm illustrate a clipping of a line segment joining two end points A(-1,5) B(3,8) by considering Clipping window with left corner at (-3,1) and upper right corner at (2,6)

→




DOSBox 0.72, Cpu Cycles: max, Frameskip 0, Program: TC

Date : / /
Page No.:

14. Program to Clip Polygon using Sutherland hodgeman polygon Clipping algorithm

* Procedure :

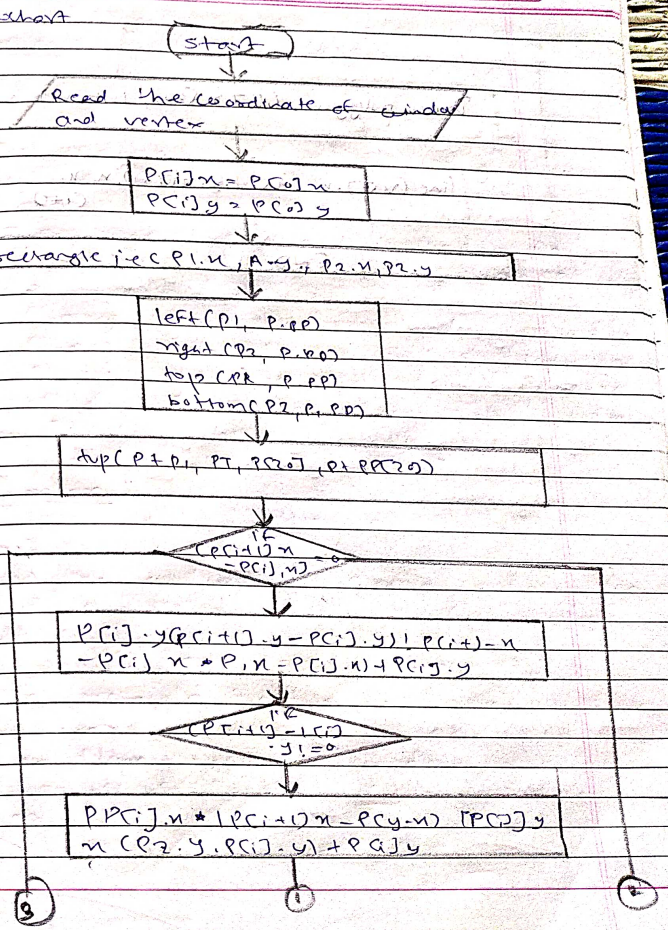
1] This algorithm performs a clipping of a polygon against each window edge in turn.
2] Every edge of polygon is compared with Clipping plane to find out the new vertex called output vertex.
3] Clipping against the left side of Clip window
4] Clipping against the top side of Clip window
5] Clipping against the right side of Clip window
6] Clipping against the bottom side of Clip window
7] There are the types of Clipping.

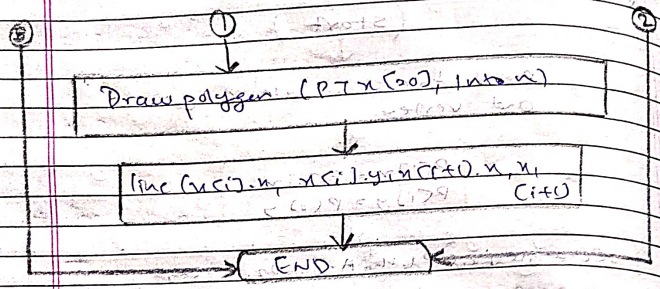
* Algorithm :

1] Start
2] Read the coordinates of clipping window then read the vertex coordinate of Clipping window.
3] Start with P[i]x = P[2]x and P[i]y = P[0].y
4] Use rectangle function to display window on screen
5] Then draw polygon on window with the help of left (P1, P, pp) right (P2, P, pp)
6] Then call left function
left (P2, P1, P1, P[20] ; P1 pp[20])

7) IF (P[i].x - P[i].x - P[i]*x) (P1.x P[i].x)
8) Then call right function
   right (P1, P2, P+P[20], P1 PP)
9) IF (P[i+1].x - P[i].x, i=0 | P[i+1] - x - P[i].x)
   (P2.x - P[i] + P[i].y)
10) Then call the dup function to clip to side of polygon.
11) IF (P[i+1].y - P[i].y, i=0) then calculate
   (P1.y - P[i].y) + P[i].x,
12) Then bottom
13) IF (P[i+1].y - P[i].y, i=0) Then calculate
   PP[i].x = (P[i+1].d - P[i].x) (P[i+1].y - P[i].y
   * (P2 * y) . P[i].y)
14) Draw polygon using draw polygon
15) END

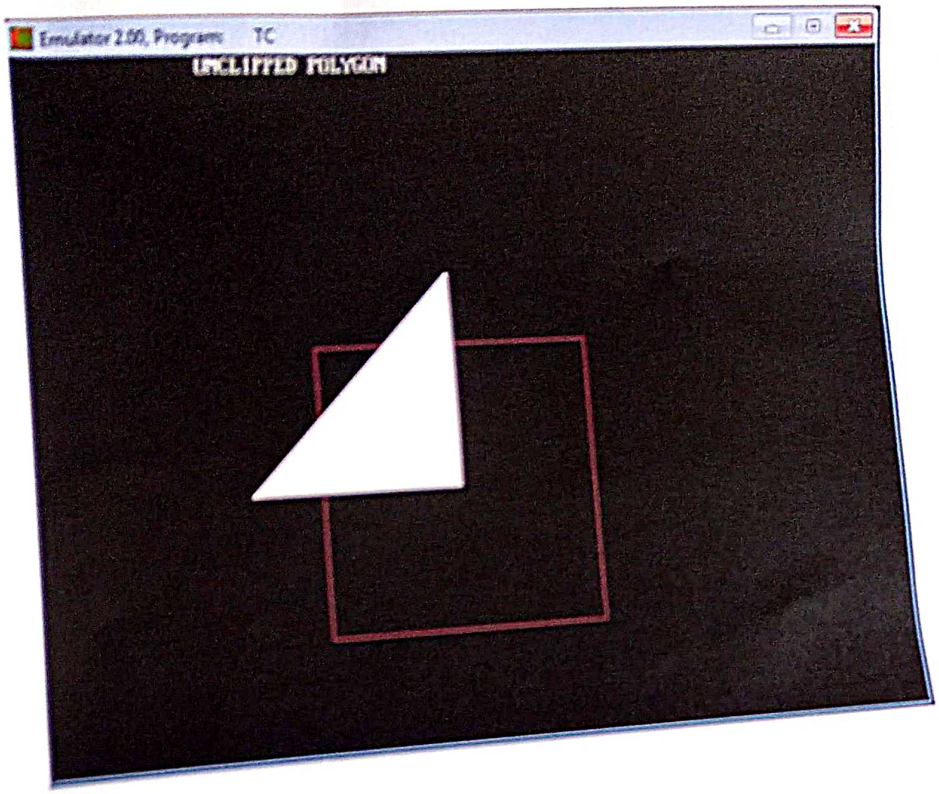
Flowchart

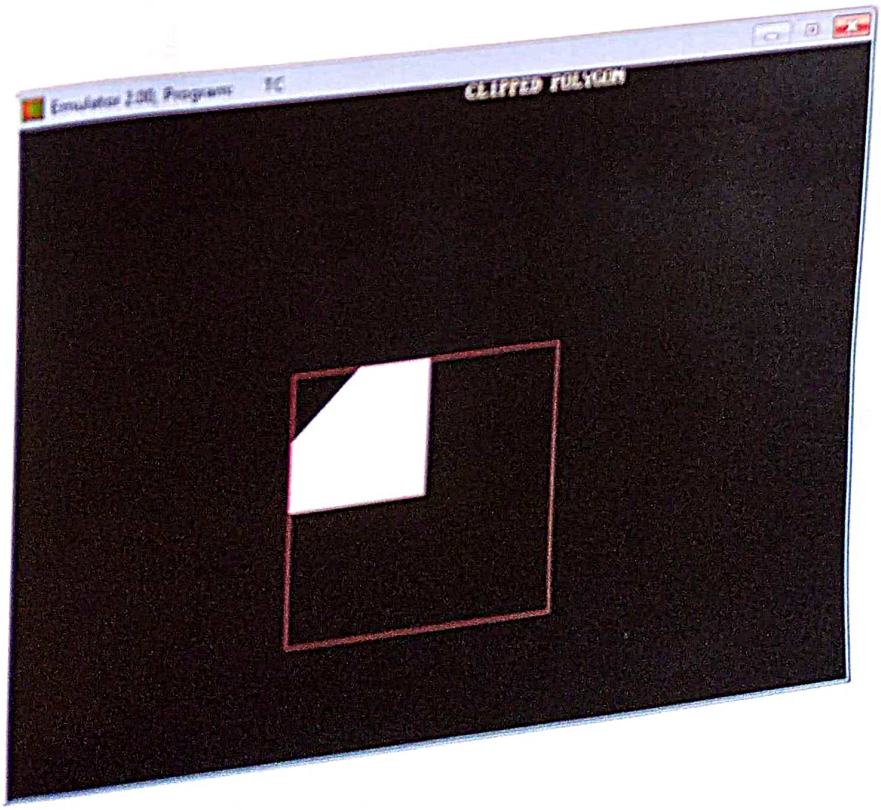





```
//14 - code
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void dda(int,int,int,int,int,int,int,int);
void dda(int x1,int y1,int x2,int y2,int xmin,int ymin,int xmax,int ymax)
{
        float dx,dy;
        float steps,x=x1,y=y1;
        int i=0;
        dx=x2-x1;
        dy=y2-y1;
        if(abs(dx)>=abs(dy))
                steps=abs(dx);
        else
                steps=abs(dy);
        dx=dx/steps;
        dy=dy/steps;
        while(i++<=steps)
        {
        if(x>=xmin && x<=xmax && y>=ymin && y<=ymax)
        {
                line(x,y,x2,y2);
                return;
        }
                x=x+dx;
                y=y+dy;
        }
}
void main()
{
        int n,gd,gm,x1,x2,y1,y2,xRec,yRec,b1,b2,b3,b4,l,b,yMin,yMax,xMin,xMax;
        float m;
        int a[10][4],i,j, flag=0,in=0;
        gd=DETECT;
        gm=DETECT;
        initgraph(&gm,&gd,"C:\\TC\\BGI");

        printf("Enter the length and breadth of the clipping window:\n");
        scanf("%d%d",&l,&b);

        printf("Enter the starting co-ord of the rectangle\n");
        scanf("%d%d",&xRec,&yRec);
        clrscr();

        rectangle(xRec,yRec,xRec+l,yRec+b);
```

```
getch();
clrscr();


printf("Enter the no of lines\n");
scanf("%d",&n);

for(i=0;i<n;i++)
{
        printf("Enter the co-ord of line %d\n",i+1);
        for(j=0;j<2;j++)
        {
                scanf("%d",&a[i][j]);
        }
        a[i][2]=a[0][0];
        a[i][3]=a[0][1];
        if(i!=0)
        {
             a[i-1][2]=a[i][0];
             a[i-1][3]=a[i][1];
        }
}
clrscr();
rectangle(xRec,yRec,xRec+l,yRec+b);
for(i=0;i<n;i++)
{
         line(a[i][0],a[i][1],a[i][2],a[i][3]);
}
getch();
clrscr();
rectangle(xRec,yRec,xRec+l,yRec+b);

xMin=xRec;
yMin=yRec;
xMax=xRec+l;
yMax=yRec+b;

for(i=0;i<n;i++)
{
        flag=0;
        x1=a[i][0];
        x2=a[i][2];
        y1=a[i][1];
        y2=a[i][3];

        if(x1>=xMin && x1<=xMax && y1>=yMin && y1<=yMax)
                flag++;

                if(x2>=xMin && x2<=xMax && y2>=yMin && y2<=yMax)
                        flag++;


                switch(flag)
                {
                        case 0:
                                break;
                        case 1: if(x2>=xMin && x2<=xMax && y2>=yMin && y2<=yMax)
                                {
                                        dda(x1,y1,x2,y2,xMin,yMin,xMax,yMax);

                                }
                                else
                                {
                                        dda(x2,y2,x1,y1,xMin,yMin,xMax,yMax);

                                }
                                break;
                        case 2: line(x1,y1,x2,y2);
                                break;
                        }
        }

        getch();

        closegraph();
}
```


Emulator 2.00, Program: TC
UNCLIPPED POLYGON


Emulator 2.00, Program: TC
CLIPPED POLYGON

Date : / /
Page No.:

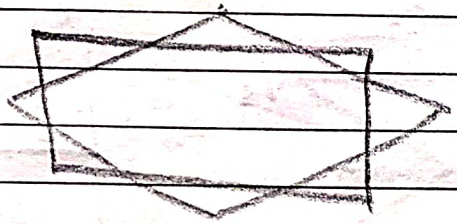
* Practical Related Question

1] IF vertex is outside the clipping window and second point is inside the clipping window, then write which points are added to output vertex list.

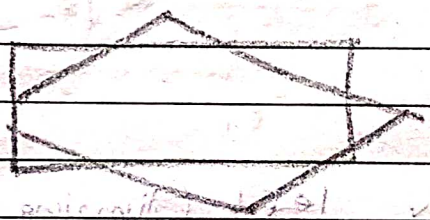
→ IF the first vertex is outside the clipping window and second point is inside the clipping window then both the intersection point of the polygon edge with window boundary and second vertex are added to o/p vertex list.

2] Write Procedure to clip polygon using sutherland Hodgeman Polygon Clipping algorithm.

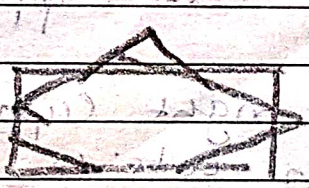
→ Before clipping :



1] After clipping against left side of window



2] After Clipping against bottom side of window :



3] After Clipping from top side of window

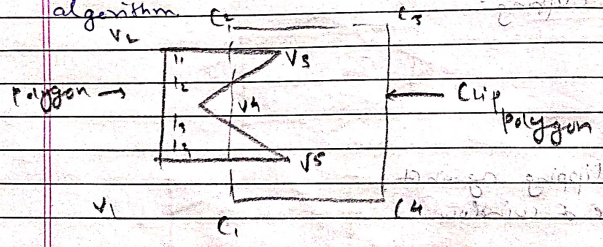
4] Polygon Clipped

* Conclusion:

Thus, we learned the program to clip polygon using Sutherland Hodgeman algorithm

* Exercise:

1] Clip the following Polygon using Cohen Sutherland Hodgeman Polygon Clipping algorithm.



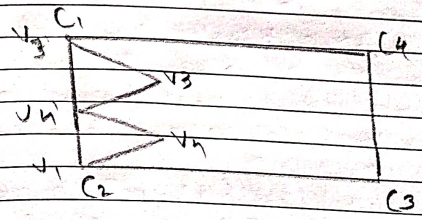
→ After left Clipping : V1', V3, V4, V2, V5'

After right Clipping :
V2 = outside V3 = inside
Thus save V2 and V3

V3 = inside V4 = outside
Save V4 and V3

V1 = outside V5 = inside
Save V1 and V5

Clipped Polygon,



Date : / /
Page No.:

15. Program to draw Hilbert's Curve

* Procedure

1] Hilbert subroutine draws the Hilbert curve
2] It takes as parameters the depth of recursion and dx and dy values that give the direction in which it should draw.
3] It recursively draws four smaller Hilbert curve and connect them with lines.

* Algorithm

1] Start
2] Initialize valid hilb (Float a, b, c, d, ex, ey, ent, hf)
3] Initialise the if condition
4] if (int > 1)
hilb (a, b, c, d, tx[a], ty[a], ent-1, hf/2)
hilb (a, b, c, d, tx[b], ty[b], ent-1, hf/2)
hilb (c, b, a, d, tx[d], ty[d], ent-1, hf/2)
2] then, in main function use do while loop and use switch case.
5] For drawing hilbert curve read the limit of window
6] Enter no of iteration the hilbert curve is generated.
7] End

```
// 15 - code
#include <stdio.h>
#define N 32
#define K 3
#define MAX N * K

typedef struct { int x; int y; } point;

void rot(int n, point *p, int rx, int ry) {
   int t;
   if (!ry) {
      if (rx == 1) {
         p->x = n - 1 - p->x;
         p->y = n - 1 - p->y;
      }
      t = p->x;
      p->x = p->y;
      p->y = t;
   }
}
void d2pt(int n, int d, point *p) {
   int s = 1, t = d, rx, ry;
   p->x = 0;
   p->y = 0;
   while (s < n) {
      rx = 1 & (t / 2);
      ry = 1 & (t ^ rx);
      rot(s, p, rx, ry);
      p->x += s * rx;
      p->y += s * ry;
      t /= 4;
      s *= 2;
   }
}
int main() {
  int d, x, y, cx, cy, px, py;
  char pts[MAX][MAX];
  point curr, prev;
  for (x = 0; x < MAX; ++x)
     for (y = 0; y < MAX; ++y) pts[x][y] = ' ';
  prev.x = prev.y = 0;
  pts[0][0] = '.';
  for (d = 1; d < N * N; ++d) {
     d2pt(N, d, &curr);
     cx = curr.x * K;
     cy = curr.y * K;
     px = prev.x * K;
```

```
        py = prev.y * K;
        pts[cx][cy] = '.';
        if (cx == px ) {
           if (py < cy)
              for (y = py + 1; y < cy; ++y) pts[cx][y] = '|';
           else
              for (y = cy + 1; y < py; ++y) pts[cx][y] = '|';
        }
        else {
           if (px < cx)
              for (x = px + 1; x < cx; ++x) pts[x][cy] = '_';
           else
              for (x = cx + 1; x < px; ++x) pts[x][cy] = '_';
        }
        prev = curr;
     }
     for (x = 0; x < MAX; ++x) {
        for (y = 0; y < MAX; ++y) printf("%c", pts[y][x]);
        printf("\n");
     }
     return 0;
}
```

Practical Related Question

1) Define curve.

→ A curve is an infinitely large set of points. Each points has two neighbour except endpoint. Curve can be broadly classified into three categories.

2) Write topological and fractal dimension of hilbert curve.

→ Topological dimension is 1 and fractal dimension is 2 of hilbert curve.

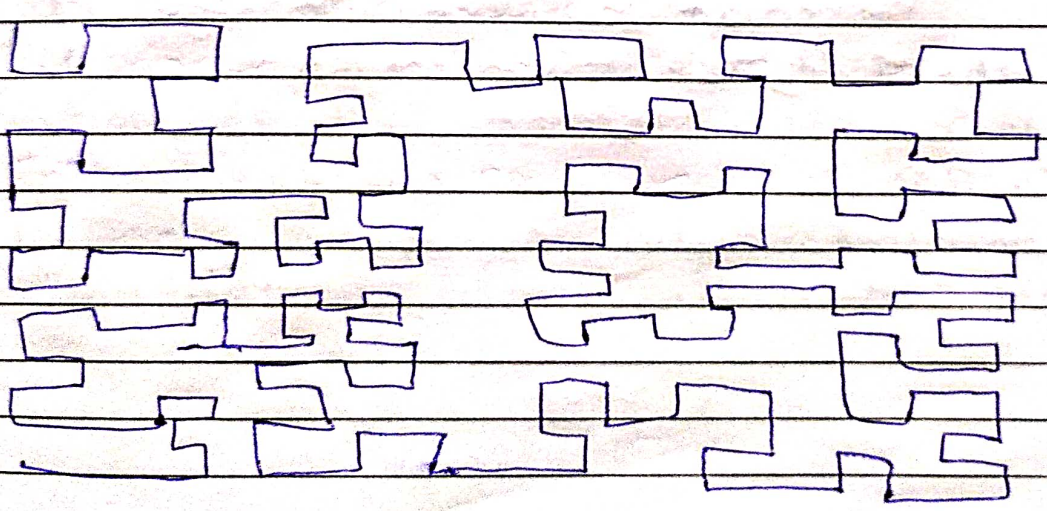
Conclusion

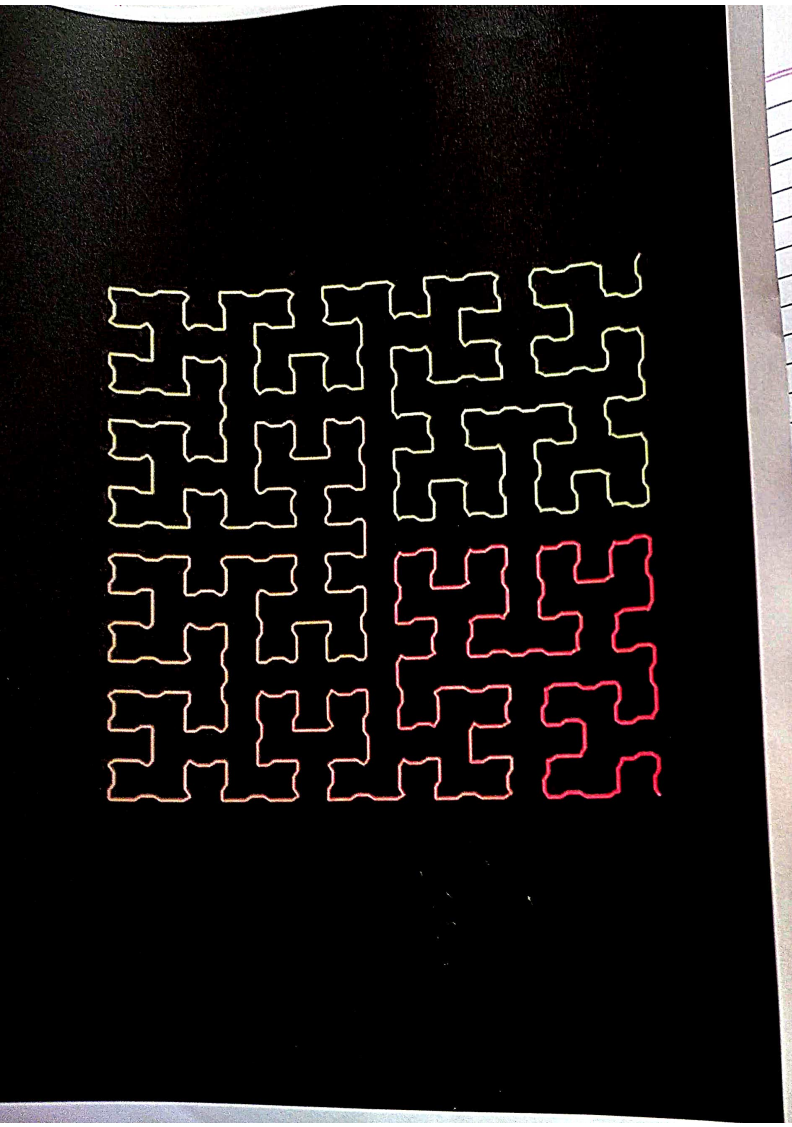
Thus, we learned to draw hilbert curve.

Exercise

1) Draw the hilbert curve

eg)





Date : / /
Page No.:

16. Program to draw Koch Curve and Bezier Curve

* Procedure

A) Koch curve:

1] Start with a line

2] Trisect the line into 3

3] Form an equilateral triangle rising out of the middle segment

4] Repeat.

B) Bezier Curve:

To each set of four point P_0, P_1, P_2, P_3, we associate a curve with the following properties.

i] It starts at P_0 and ends at P_3

ii] When it starts from P_0 it heads directly towards P_1, and when it arrives at P_3 it is coming from the direction of P_2

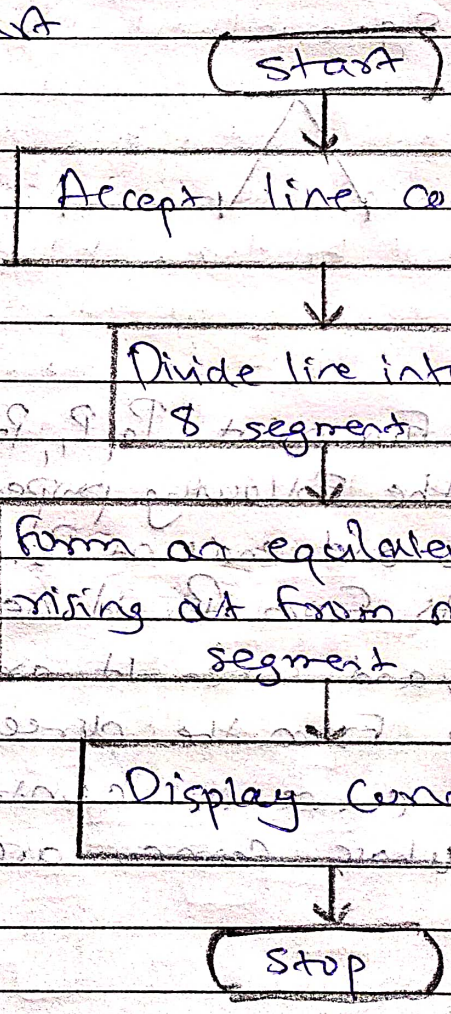
iii] The entire curve is contained in the quadrilateral whose corners are the four given points.

Date :
Page No.:

* Algorithm

1] Start
2] Accept line coordinate
3] Divide line into 3 coordinate
4] Form a equilateral triangle rising out From middle segment.
5] Repeat with newly formed segment.
6] Display Curve
7] stop

* Flowchart



```
//18 - 1 - koch
#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,i,n, x1 = 100, y1 = 100, x2 = 400, y2 = 400;
 initgraph(&gd,&gm,"C://TC3//BGI");
 printf("Enter number of interations");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
 cleardevice();
 koch(x1, y1, x2, y2, i);
 getch();
 }
 return 0;
```

```
//16 - 1 - koch
#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,i,n, x1 = 100, y1 = 100, x2 = 400, y2 = 400;
 initgraph(&gd,&gm,"C://TC3//BGI");
 printf("Enter number of interations");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
 cleardevice();
 koch(x1, y1, x2, y2, i);
 getch();
 }
 return 0;
```

```
//16-2-code
#include<graphics.h>
#include<math.h>
#include<conio.h>
#include<stdio.h>
void main()
{
int x[4],y[4],i;
double put_x,put_y,t;
int gr=DETECT,gm;
initgraph(&gr,&gm,"C:\\TURBOC3\\BGI");
printf("\n****** Bezier Curve ***********");
printf("\n Please enter x and y coordinates ");
for(i=0;i<4;i++)
{
scanf("%d%d",&x[i],&y[i]);
putpixel(x[i],y[i],3);              // Control Points
}

for(t=0.0;t<=1.0;t=t+0.001)          // t always lies between 0 and 1
{
put_x = pow(1-t,3)*x[0] + 3*t*pow(1-t,2)*x[1] + 3*t*t*(1-t)*x[2] + pow(t,3)*x[3]; // Formula to
draw curve
put_y =  pow(1-t,3)*y[0] + 3*t*pow(1-t,2)*y[1] + 3*t*t*(1-t)*y[2] + pow(t,3)*y[3];
putpixel(put_x,put_y, WHITE);          // putting pixel
}
getch();
closegraph();
}
```

Date : / /
Page No.:

Practical Related Question

1] Define fractal.

→

1] Fractals are complex pictures generated by a computer from a single formula.
2] They are created using repeatative step this mean one formula is repeated again and again with slight validation in value.

2] Define fractal dimension and topological dimension.

→

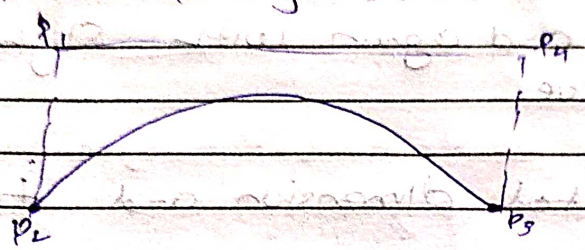
1] The objects which are having smooth surface and regular shape are generally described by using equation.
2] But natural object such as mountains tree, oceans have irregular shape
3] It will be very difficult to draw those shapes by using the equation.
4] Fractal is the most interesting form a mathematical perspective.
5] So we can describe natural objects by using fractals.

* Conclusion

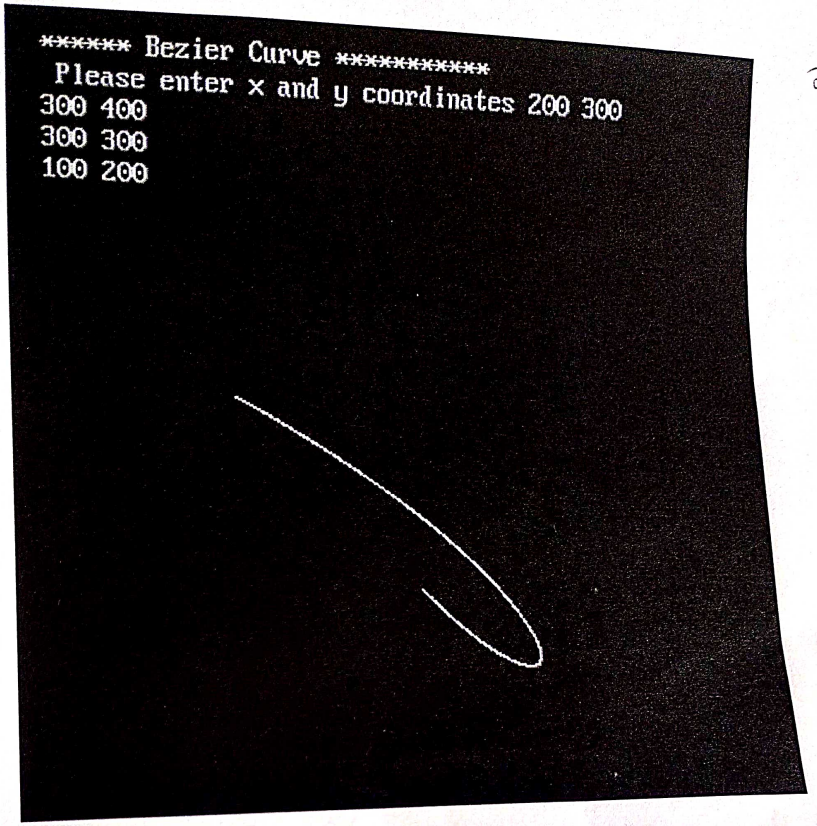
Thus we performed the program to form Koch curve and bezier curve

* Exercise

1) Draw the following Bezier curve



2) Draw the following Koch curve.


****** Bezier Curve **********
Please enter x and y coordinates 200 300
300 400
300 300
100 200