

Navarachana University-School of Engineering and Technology

**INDEX**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Name: SAHIL PANDYA** | | | | | | | | **Enrollment No.: 21124097** | | | |
| **Subject code** | | **:** | CS241 | | **Semester** | **:** | 4th | **Academic Year** | **:** | 2023 | |
| **Subject name** | | **:** | COMPUTER GRAPHICS LABORATORY | | | | | | | | |
| **Sr.**  **No.** | | **Date of experiment** | | | **Name of Experiment** | | | | | | | **Signature** | |
| 01 | |  | | | Study different Graphics commands from Graphics.h header file | | | | | | |  | |
| 02 | |  | | | Implement the following shapes using graphics functions   1. Snowman 2. Human Face 3. Smiley 4. Car 5. Line with different styles | | | | | | |  | |
| 03 | |  | | | Implement Digital Differential Analyser scan conversion algorithm (DDA). | | | | | | |  | |
| 04 | |  | | | Implement Bresenham’s line drawing scan conversion algorithm. | | | | | | |  | |
| 05 | |  | | | Implement Mid-Point Circle Drawing scan conversion algorithm. | | | | | | |  | |
| 06 | |  | | | Implement Mid-Point ellipse drawing scan conversion algorithm. | | | | | | |  | |
| 07 | |  | | | Implement Flood fill Algorithm. | | | | | | |  | |
| 08 | |  | | | Implement Boundary fill Algorithm. | | | | | | |  | |
| 09 | |  | | | Write an interactive Program to achieve all 2D transformations. (i.e. Translation, Rotation, Scaling, Reflection, Shearing) | | | | | | |  | |
| 10 | |  | | | Implement Cohen Sutherland Line Clipping algorithm. | | | | | | |  | |
| 11 | |  | | | Implement the shapes using turtle programming | | | | | | |  | |

## **PRACTICAL 1**

**Aim: Study and Implement various functions of Graphics functions All the graphics functions will be written as per below format**

##### **A)**

**Function Syntax:**

FOR CREATING A LINE:

line(x cord of 1st point,y cord of 1st point,x cord of 2nd point,y cord of 2nd point);

**Function Code:** include <stdio.h> include <graphics.h>

int main()

{

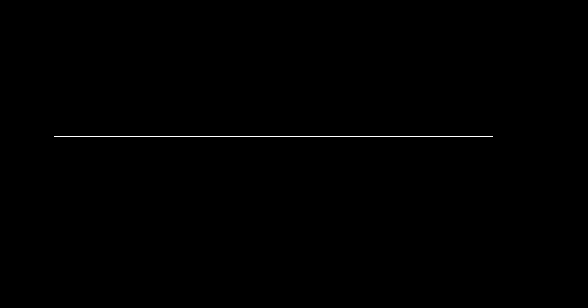
int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\TC\\BGI");

line(400,200,50,200);

return 0;

}

##### **Output:**



**B)**

##### **Function Syntax:**

FOR CREATING A ARC:

arc(x cord,y cord,startring angle,ending angle,radius)

**Function Code:** include <stdio.h> include <graphics.h>

int main()

{

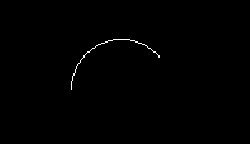
int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\TC\\BGI");

arc(100,200,40,180,40);

return 0;

}

##### **Output:**



**C)**

##### **Function Syntax:**

FOR CREATING A CIRCLE:

circle(x center,y center,radius)

**Function Code:** include <stdio.h> include <graphics.h>

int main()

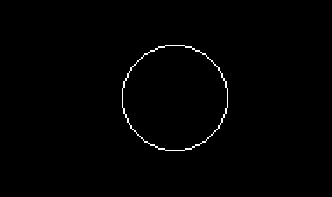
{

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

circle(200,200,30); return 0;

}

##### **Output:**



**D)**

##### **Function Syntax:**

FOR CREATING A ELLIPSE:

elipse(x cord,y cord,starting angle,ending angle,x radius,y radius)

##### **Function Code:**

include <stdio.h>

include <graphics.h> int main()

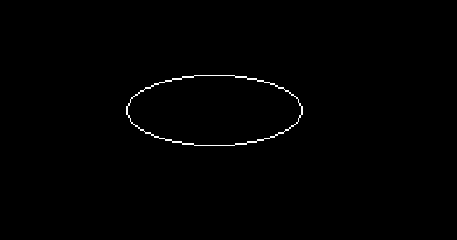
{

int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\TC\\BGI"); ellipse(150,150,0,360,50,20);

return 0;

}

##### **Output:**



**E)**

##### **Function Syntax:**

FOR CREATING A RECTANGLE:

rectangle(left top,left bottom,right top,right bottom)

##### **Function Code:**

include <stdio.h> include <graphics.h>

int main()

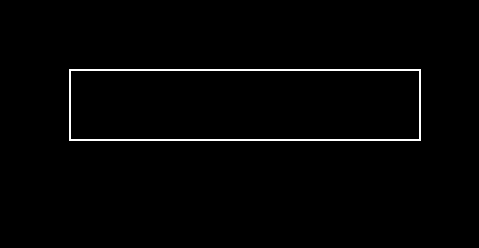
{

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

rectangle(400,40,600,80); return 0;

}

##### **Output:**



**F)**

##### **Function Syntax:**

FOR CREATING A BAR(2D):

bar(left top,left bottom,right top,right bottom)

**Function Code:** include <stdio.h> include <graphics.h>

int main()

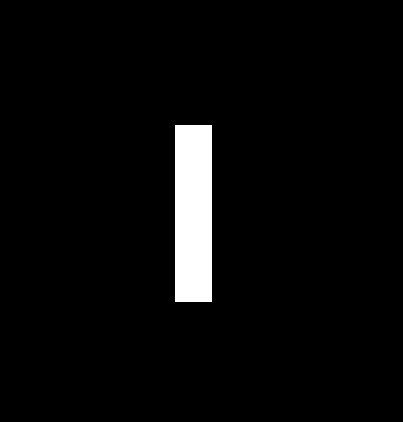
{

int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\TC\\BGI"); bar(400,100,420,200);

return 0;

}

##### **Output:**



**G)**

##### **Function Syntax:**

FOR CREATING A BAR(3D):

bar3d(left top,left bottom,right top,right bottom,depth,top)

**Function Code:** include <stdio.h> include <graphics.h>

int main()

{

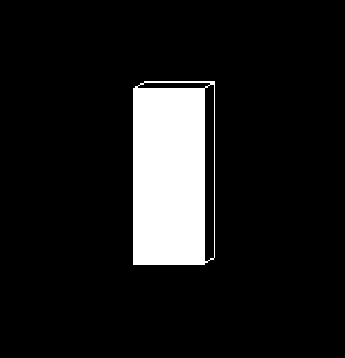
int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\TC\\BGI");

bar3d(280,200,320,300,6,6);

return 0;

}

##### **Output:**

  
**H)**

##### **Function Syntax:**

FOR CREATING A PIESLICE:

pieslice(x cord,y cord,start angle,ending angle,radius)

**Function Code:** include <stdio.h> include <graphics.h>

int main()

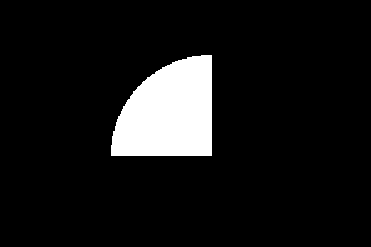
{

int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\TC\\BGI"); pieslice(200,200,90,180,80);

return 0;

}

##### **Output:**



**I)**

##### **Function Syntax:**

TO COLOR AN OBJECT BOUNDARY:

Setcolor (colornumber)

**Function Code:** include <stdio.h> include <graphics.h>

int main()

{

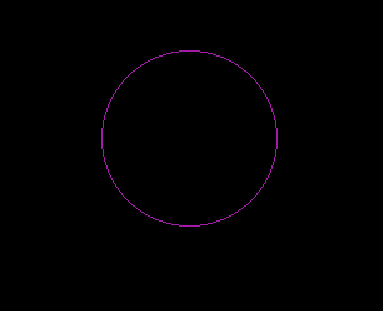
int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TC\\BGI"); setcolor(5);

circle(200,200,70); return 0;

}

##### **Output:**



**J)**

##### **Function Syntax:**

TO COLOR BACKGROUND:

setbkcolor(colornumber)

**Function Code:** include <stdio.h> include <graphics.h>

int main()

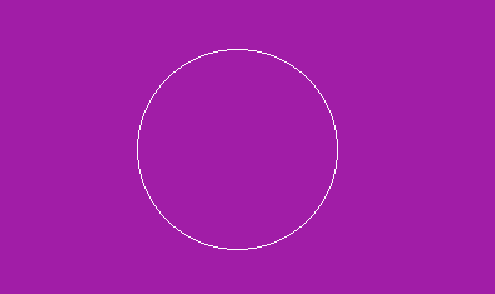
{

int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\TC\\BGI"); setbkcolor(5);

circle(200,200,80); return 0;

}

##### **Output:**



**K)**

##### **Function Syntax:**

TO SET LINE STYLE:

setlinestyle(type,depth)

**Function Code:** include <stdio.h> include <graphics.h>

int main()

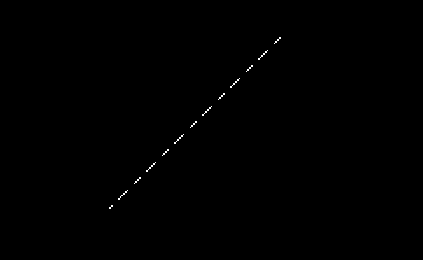
{

int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\TC\\BGI"); setlinestyle(CENTER\_LINE,3,4); line(200,100,100,200);

return 0;

}

##### **Output:**



**L)**

##### **Function Syntax:**

TO SET TEXT STYLE: settextstyle(FONT,DIRECTION,CHARACTER SIZE)

**Function Code:** include <stdio.h> include <graphics.h>

int main()

{

int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\TC\\BGI"); settextstyle(BOLD\_FONT,0,10); outtext("NUV");

return 0;

}

##### **Output:**

  
**M)**

##### **Function Syntax:**

setfillstyle(HATCH\_FILL,GREEN);

**Function Code:** include <stdio.h> include <graphics.h>

void main()

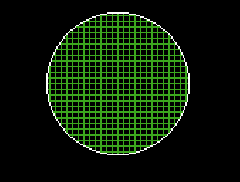
{

int graphic\_driver = DETECT, graphic\_mode; initgraph(&graphic\_driver, &graphic\_mode, "C://TC//BGI"); setfillstyle(HATCH\_FILL, GREEN);

circle(100,100,50); floodfill(101,101,WHITE);

}

**Output:**



**Conclusion:**

We create basic different shapes by using graphic functions like circles, ellipse, pie slice, etc. and can format text by using textstyle function.

## **PRACTICAL 2**

**Aim: Write a program to implement the graphics mentioned graphics**

##### **Snowman Code:**

include <stdio.h> include <graphics.h>

int main()

{

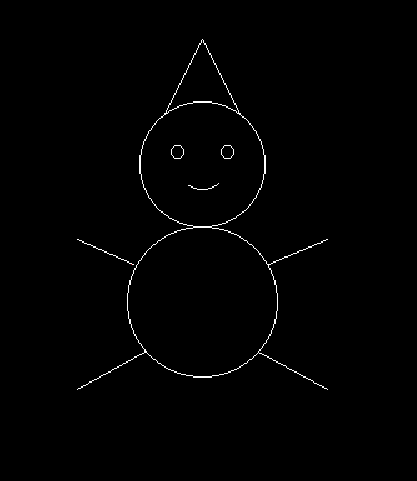
int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\TC\\BGI"); setcolor(WHITE); circle(200,200,50); circle(180,190,5);

circle(220,190,5); arc(200,200,235,315,20); line(170,160,200,100); line(200,100,230,160);

circle(200,310,60); line(100,260,145,280); line(300,260,253,280); line(154,350,100,380); line(300,380,245,350);

}

##### **Output:**

  
**Smiley Code:**

include <stdio.h> include<graphics.h>

int main()

{

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

//head circle(200,200,70);

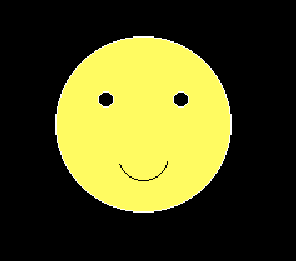
setfillstyle(SOLID\_FILL, YELLOW); circle(170,180,6);

circle(230,180,6); floodfill(200, 200, WHITE); setcolor(BLACK); arc(200,225,200,350,20);

return 0;

}

##### **Output:**



1. **Line with Different Styles Code:**

include <stdio.h> include <graphics.h>

int main()

{ int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\TC\\BGI"); setlinestyle(SOLID\_LINE,3,1); line(400,50,200,50);

setlinestyle(CENTER\_LINE,3,1); line(400,100,200,100);

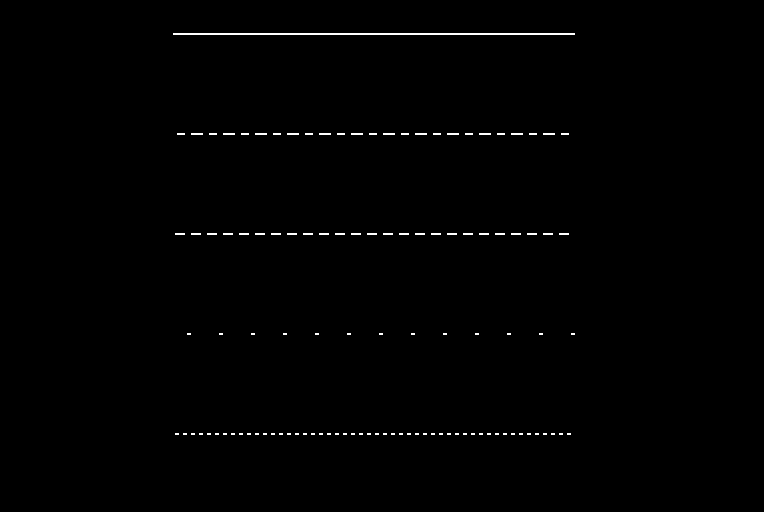
setlinestyle(DASHED\_LINE,3,1); line(400,150,200,150);

setlinestyle(USERBIT\_LINE,3,1); line(400,200,200,200);

setlinestyle(DOTTED\_LINE,3,1); line(400,250,200,250);

}

##### **Output:**



1. **Pie Chart Code:**

include <stdio.h>

include<graphics.h>

int main()

{

int gd= DETECT,gm; initgraph(&gd,&gm,"C:\\TC\\BGI"); setfillstyle(SOLID\_FILL, YELLOW); pieslice(130,100,0,60,50); setfillstyle(HATCH\_FILL, GREEN); pieslice(130,100,60,160,50);

setfillstyle(XHATCH\_FILL, RED); pieslice(130,100,160,200,50);

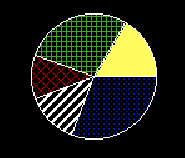
setfillstyle(SLASH\_FILL, WHITE); pieslice(130,100,200,250,50);

setfillstyle(CLOSE\_DOT\_FILL, BLUE); pieslice(130,100,250,360,50);

return 0;

}

##### **Output:**



1. **Bar Chart Code:**

include <stdio.h> include <graphics.h>

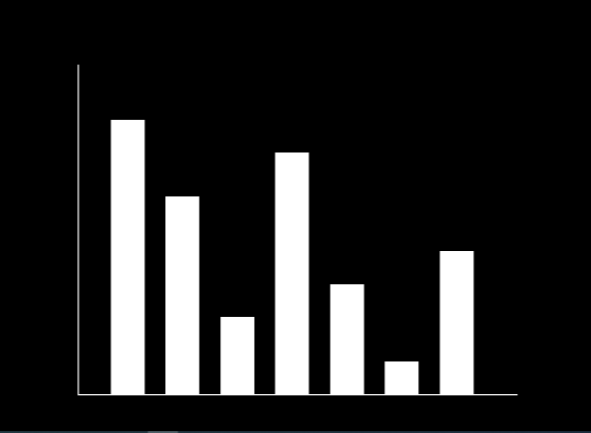
int main()

{

int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\TC\\BGI"); line(100,100,100,400); line(100,400,500,400); bar(130,150,160,400); bar(180,220,210,400); bar(230,330,260,400); bar(280,180,310,400); bar(330,300,360,400); bar(380,370,410,400); bar(430,270,460,400);

}

##### **Output:**



1. **Concentric Circle Code:**

include <stdio.h> include <graphics.h>

int main()

{

int gd=DETECT,gm,i; initgraph(&gd,&gm,"C:\\TC\\BGI"); for(i=60;i>2;i--)

{

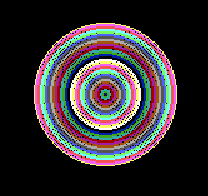
setcolor(i/2); circle(200,200,i-3);

}

return 0;

}

##### **Output:**



**7.) HumanFace**

include<stdio.h>

include<conio.h>

include<graphics.h>

void main()

{

int gdriver=DETECT,gmode=0; int I;

initgraph(&gdriver,&gmode,"C:\\TC\\BGI"); cleardevice();

rectangle(0,0,639,479); arc(320,230,190,180,120); line(210,190,430,190); arc(320,230,160,380,120); arc(440,230,270,450,20); arc(200,230,90,270,20);

circle(260,220,10) circle(380,220,10); for(I=0;I<2;I++)

{ arc(260,213+I,30,150,13); arc(380,213+I,30,150,13);

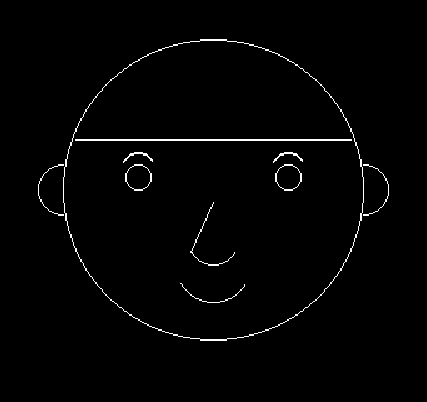
} line(320,240,302,280);

;

getch();

}

**Output:**



**Conclusion:**

We can use various functions of graphics.h like line(), arc(), ellipse(), circle() etc. to draw real life object.

# **PRACTICAL - 3**

**Aim**: Write a program to implement DDA (Digital Differential Analyzer) Line Drawing Algorithm.

**Algorithm:**

**Step 1: Start Algorithm**

**Step 2: Declare x1,y1,x2,y2,dx,dy,x,y as integer variables. Step 3: Enter value of x1,y1,x2,y2.**

**Step 4: Calculate dx = x2-x1 Step 5: Calculate dy = y2-y1 Step 6: If ABS (dx) > ABS (dy)**

**Then step = abs (dx)**

**Else**

**Step 7: xinc=dx/step**

**yinc=dy/step assign x = x1 assign y = y1**

**Step 8: putpixel (x, y)**

**Step 9: x = x + xinc**

**y= y + yinc putpixel (x, y)**

**Step 10: Repeat step 9 until x = x2 Step 11: End Algorithm**

### **Code:**

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

void main()

{

int graphicDriver = DETECT ,graphicMode, i;

float x, y, dx, dy, steps; int x1, x2, y1, y2;

initgraph(&graphicDriver, &graphicMode, "C:\\TC\\BGI"); setbkcolor(WHITE);

x1 = 100 , y1 = 200, x2 = 400, y2 = 200;

dx = x2 - x1; dy = y2 - y1; if(dx>=dy)

{

steps = dx;

}

else

{

steps = dy;

}

dx = dx/steps; dy = dy/steps; x = x1;

y = y1; i = 1;

while(i<= steps)

{

putpixel(x, y, BLUE); x += dx;

y += dy; i=i+1;

}

getch(); closegraph();

}

### **Output:**



**Conclusion:**

Using DDA algorithm we can easily create lines as DDA algorithm is the simplest line drawing algorithm.

# **PRACTICAL – 4**

**Aim**: Write a program to implement Bressenham’s Line Drawing Algorithm.

**Algorithm:**

**Step 1: Start.**

**Step 2: Now, we consider Starting point as (x1, y1) and endingpoint (x2, y2). Step 3: Now, we have to calculate dx and dy.**

**x = x2-x1 y = y2-y1 m = dy/dx**

**Step 4: Now, we will calculate the decision parameter pk with following formula.**

**pk = 2dy-dx**

**Step 5: The initial coordinates of the line are (xk, yk), and the next coordinatesare (xk+1, yk+1). Now, we are going to calculate two cases for decision parameter pk**

**Step 6: We will repeat step 5 until we found the ending point of the line and the total number of iterations =dx-1.**

**Step 7: Stop.**

### **Code:**

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

void main()

{

int graphicDriver = DETECT, graphicMode; int x1, x2, y1, y2;

int dx, dy, p, x, y;

initgraph(&graphicDriver, &graphicMode, "C:\\TC\\BGI"); setbkcolor(WHITE);

x1 = 100, y1 = 100, x2 = 400, y2 = 100;

dx=x2-x1; dy=y2-y1;

x=x1; y=y1;

p=2\*dy-dx;

while(x<x2)

{

if(p>=0)

**{**

putpixel(x,y,RED); y=y+1;

p=p+2\*dy-2\*dx;

}

else

{

putpixel(x,y,RED); p=p+2\*dy;

}

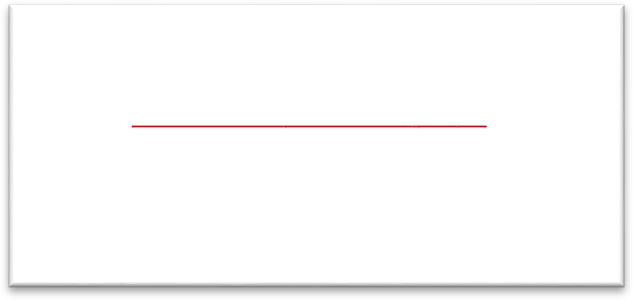
x=x+1;

}

getch(); closegraph();

}

### **Output:**



**Conclusion:**

In Bresenhams Algorithm uses simple calculation like addition and subtraction for creating a line.

### **Practical – 5**

**AIM- Implement Mid-Point Circle Drawing scan conversion algorithm.**

**CODE:**

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

int main()

{

int p,x,y,r=80;

void cal\_point(int xc,int yc,int x,int y); int gdriver = DETECT, gmode;

initgraph(&gdriver, &gmode, "c:\\tc\\bgi"); x=0;

y=r; p=1-r;

while(x<y)

{

if(p<0)

{

x=x+1; y=y;

p = p+2\*x+1;

}

else

{

x=x+1; y=y-1;

p = p+2\*x+1-2\*y;

}

cal\_point(100,100,x,y);

}

getch(); closegraph(); return 0;

}

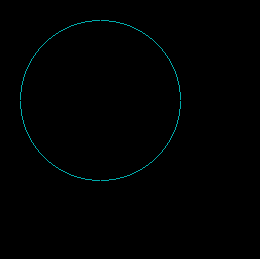
void cal\_point(int xc,int yc,int x,int y)

{

putpixel(xc+x,yc+y,3); putpixel(xc-x,yc-y,3); putpixel(xc+x,yc-y,3); putpixel(xc-x,yc+y,3); putpixel(xc+y,yc+x,3); putpixel(xc-y,yc-x,3); putpixel(xc+y,yc-x,3); putpixel(xc-y,yc+x,3);

}

**OUTPUT:**



### **PRACTICAL-6**

**AIM: Implement Mid-Point ellipse drawing scan conversion algorithm.**

**CODE:**

include<stdio.h> include<graphics.h>

void main() {

long x,y,x\_center,y\_center;

long a\_sqr,b\_sqr, fx,fy, d,a,b, tmp1, tmp2; int g\_driver=DETECT,g\_mode;

clrscr();

initgraph(&g\_driver, &g\_mode,"C:\\TC\\BGI"); printf ("\*\*\* MID POINT ELLIPSE ALGORITHM \*\*\*");

printf("\n\n Enter coordinate x and y = "); scanf ("%1d%1d", &x\_center, &y\_center); printf("\n Now enter constants a and b = "); scanf ("%1d%1d", &a, &b);

x=0;

y=b; a\_sqr=a\*a; b\_sqr=b\*b; fx=2\*b\_sqr\*x; fy=2\*a\_sqr\*y;

d=b\_sqr-(a\_sqr\*b) +(a\_sqr \* 0.25); do

{

putpixel(x\_center+x, y\_center+y,1); putpixel(x\_center-x, y\_center-y,1); putpixel(x\_center+x, y\_center-y,1); putpixel(x\_center-x, y\_center+y,1);

if(d<0)

{

d=d+fx+b\_sqr;

}

else

{

y=y-1;

d=d+fx+-fy+b\_sqr; fy=fy - (2\*a\_sqr);

}

x=x+1;

fx=fx + (2\*b\_sqr); delay (10);

}

while (fx<fy);

tmp1=(x + 0.5)\*(x + 0.5);

tmp2=(y - 1) \* (y - 1); d=b\_sqr\*tmp1+a\_sqr\*tmp2-(a\_sqr\*b\_sqr); do

{

putpixel(x\_center+x, y\_center+y,1); putpixel(x\_center-x, y\_center-y,1); putpixel(x\_center+x, y\_center-y,1); putpixel(x\_center-x, y\_center+y,1);

if(d>=0)

d=d-fy+a\_sqr; else

{

x=x + 1;

d=d+fx-fy+a\_sqr; fx=fx + (2\*b\_sqr);

}

y=y - 1;

fy=fy-(2\*a\_sqr);

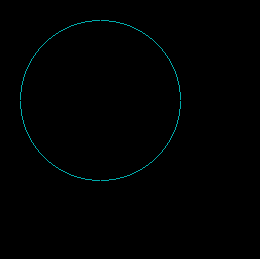
}

while (y>0);

getch(); closegraph();

}

OUTPUT:



# **PRACTICAL 7**

**Aim: Write a program to Implement the Flood Fill Algorithms**

**Algorithm:**

**Procedure floodfill (x, y,fill\_ color, old\_color: integer)**

**{**

**If (getpixel (x, y)==old\_color)**

**{**

**setpixel (x, y, fill\_color);**

**floodfill (x+1, y, fill\_color, old\_color); floodfill (x-1, y, fill\_color, old\_color); flloodfill (x, y+1, fill\_color, old\_color); floodfill (x, y-1, fill\_color, old\_color);**

**}**

**}**

### **Code:**

include<stdio.h> include<conio.h> include<graphics.h> include<dos.h>

void flood(int,int,int,int); void main()

{

int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\TC\\BGI"); rectangle(50,50,100,100); flood(51,51,3,0);

getch();

}

void flood(int x,int y,int fillColor, int oldColor)

{

if(getpixel(x,y)==oldColor)

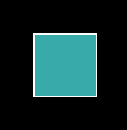
{

delay(2); putpixel(x,y,fillColor); flood(x+1,y,fillColor,oldColor); flood(x-1,y,fillColor,oldColor); flood(x,y+1,fillColor,oldColor); flood(x,y-1,fillColor,oldColor);

}

}

### **Output:**



**Conclusion:**

In fill algorithm, we start from a specified interior point (x, y) and reassign all pixel values are currently set to a given interior color with the desired color. Using either a 4-connected or 8-connected approaches, we then step through pixel positions until all interior points have been repainted.

# **PRACTICAL-8**

**Aim: Write a program to Implement the Boundary Fill**

**Algorithm:**

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);

}

}

### **Code:**

include <graphics.h>

void boundaryFill8(int x, int y, int fill\_color,int boundary\_color)

{

if(getpixel(x, y) != boundary\_color && getpixel(x, y) != fill\_color)

{

delay(1);

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

rectangle(50, 50, 100, 100);

boundaryFill8(51, 51, 5, 15); getch();

}

### **Output:**



**Conclusion:**

Boundary fill algorithm starts at a pixel inside the polygon to be filled and paints the interior proceeding outwards towards the boundary. This algorithm works only if the color with which the regio has to be filled and the color of the boundary of the region are different. If the boundary is of one single color, this approach proceeds outwards pixel by pixel until it hits the boundary of the region.

# **PRACTICAL – 9**

### **2D Translation Algorithm**

**Step1: Read x,y,tx,ty Step2:-x’=x+tx,y’=y+ty**

**Step3:-Print the new points of x’ and y’**

### **Code:**

include <stdio.h> include <graphics.h>

int main()

{

int gd=DETECT,gm,x1=100,x2=150,x3=200,y1=150,y2=200,y3=200,tx=100,ty=50,sx=3

,sy=2;

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

line(x1,y1,x2,y2);

line(x2,y2,x3,y3);

line(x3,y3,x1,y1);

//TRANSLATE

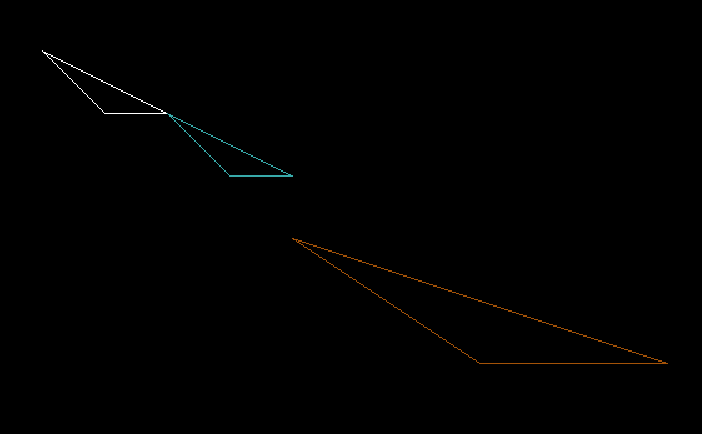
setcolor(3); line(x1+tx,y1+ty,x2+tx,y2+ty); line(x2+tx,y2+ty,x3+tx,y3+ty); line(x3+tx,y3+ty,x1+tx,y1+ty);

//SCALE

setcolor(6); line(x1\*sx,y1\*sy,x2\*sx,y2\*sy); line(x2\*sx,y2\*sy,x3\*sx,y3\*sy); line(x3\*sx,y3\*sy,x1\*sx,y1\*sy);

}

### **Output:**



**2D Scaling Algorithm:**

**Step1:Read Sx,Sy Step2:x’=x\*Sx,y’=y\*Sy**

**Step3:-Print the new Points of x’ and y’**

### **Code:**

include <graphics.h> int main()

{

int gd = DETECT, gm,x1 = 100, y1 = 100,x2 = 150, y2 = 150,x3 = 200, y3 = 100,sx = 2, sy = 2,

nx1 = x1 \* sx, ny1 = y1 \* sy, nx2 = x2 \* sx, ny2 = y2 \* sy,

nx3 = x3 \* sx, ny3 = y3 \* sy; initgraph(&gd, &gm, "C:\\TC\\BGI");

// draw original triangle setcolor(6);

line(x1, y1, x2, y2);

line(x2, y2, x3, y3);

line(x3, y3, x1, y1);

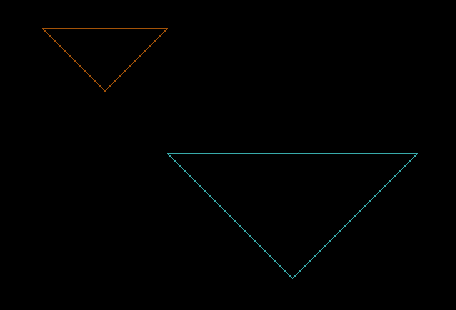
// draw scaled triangle setcolor(3);

line(nx1, ny1, nx2, ny2); line(nx2, ny2, nx3, ny3); line(nx3, ny3, nx1, ny1); getch();

closegraph(); // close graphics return 0;

}

### **Output:**



**2D Rotation Algorithm:**

**Step1:Read the x,y**

**Step 2:x’=x.cos(theta)-ysin(theta) y'=y.sin(theta)+xcos(theta)**

**Step3:Print the new values of x’ and y’**

### **Code:**

include <stdio.h> include<math.h> include <graphics.h>

int main()

{

float x1=100,x2=150,x3=200,y1=150,y2=200,y3=200,tx=380,ty=200,O=90,x11,x22,x33,y 11,y22,y33;

int gd=DETECT,gm; initgraph(&gd,&gm, "C:\\TC\\BGI"); setcolor(5);

line(x1,y1,x2,y2);

line(x2,y2,x3,y3);

line(x3,y3,x1,y1);

//TRANSLATE AND ROTATE

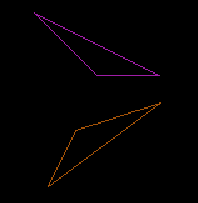
setcolor(6);

x11=(x1\*cos(O))-(y1\*sin(O))+tx; y11=(x1\*sin(O))+(y1\*cos(O))+ty;

x22=(x2\*cos(O))-(y2\*sin(O))+tx; y22=(x2\*sin(O))+(y2\*cos(O))+ty; x33=(x3\*cos(O))-(y3\*sin(O))+tx; y33=(x3\*sin(O))+(y3\*cos(O))+ty; printf("%0.2f",x33); line(x11,y11,x22,y22); line(x22,y22,x33,y33); line(x33,y33,x11,y11);

}

### **Output:**



**2D reflection Algorithm:**

**Step1: -Read x,y**

**Step2: - [ x’** **y’] = [ 1** **0] [x** **y] //Reflection with reference to X axis [0 -1]**

**[x’ y’] = [-1** **0] [x y]** **//Reflection with reference to Y axis [0** **1]**

**[x’ y’]=[-1 0 ][x** **y]** **//Reflection with reference to X-Y Axis [0** **-1]**

**Step 3: Print the New Values of x’ and y’**

### **Code:**

include <stdio.h> include<math.h> include <graphics.h>

int main()

{

int x1=100,x2=150,x3=200,y1=150,y2=200,y3=200,tx=300,ty=300,x11,x22,x33,y11,y22, y33;

int gd=DETECT,gm; initgraph(&gd,&gm, "C:\\TC\\BGI"); setcolor(6);

line(x1,y1,x2,y2);

line(x2,y2,x3,y3);

line(x3,y3,x1,y1);

//TRANSLATE AND ROTATE

setcolor(2); x11=(x1\*(-1))+tx;

y11=(y1\*(-1))+ty;

x22=(x2\*(-1))+tx;

y22=(y2\*(-1))+ty;

x33=(x3\*(-1))+tx;

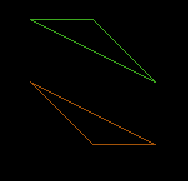
y33=(y3\*(-1))+ty;

printf("%d",x33); line(x11,y11,x22,y22);

line(x22,y22,x33,y33); line(x33,y33,x11,y11);

}

### **Output:**



**2D Shearing Code:**

include <stdio.h> include<math.h> include <graphics.h>

int main()

{

int gd=DETECT,gm,x1=100,x2=150,y1=150,y2=100,shear\_f=2,x=200,y=100; initgraph(&gd,&gm, "C:\\TC\\BGI");

setcolor(WHITE);

//TRANSLATE

setcolor(13); cleardevice(); line(x,y,x1,y1);

line(x1,y1,x2,y2);

line(x2,y2,x,y);

setcolor(2); x=x+ y\*shear\_f;

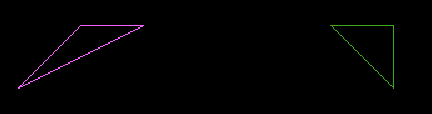
x1=x1+ y1\*shear\_f; x2=x2+ y2\*shear\_f; line(x,y,x1,y1);

line(x1,y1,x2,y2);

line(x2,y2,x,y); getch(); closegraph()

}

### **Output:**



**Conclusion:**

2D transformation can be used to reposition the graphics on the screen and change their size or orientation

### **PRACTICAL-10**

**AIM - Implement Cohen Sutherland Line Clipping algorithm.**

**CODE:**

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

void main()

{

int rcode\_begin[4]={0,0,0,0},rcode\_end[4]={0,0,0,0},region\_code[4]; int W\_xmax,W\_ymax,W\_xmin,W\_ymin,flag=0;

float slope;

int x,y,x1,y1,i, xc,yc; int gr=DETECT,gm;

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

printf("\n\*\*\* Cohen Sutherlsnd Line Clipping algorithm \*\*\*\*"); printf("\n Now, enter XMin, YMin =");

scanf("%d %d",&W\_xmin,&W\_ymin); printf("\n First enter XMax, YMax ="); scanf("%d %d",&W\_xmax,&W\_ymax);

printf("\n Please enter intial point x and y= "); scanf("%d %d",&x,&y);

printf("\n Now, enter final point x1 and y1= "); scanf("%d %d",&x1,&y1);

cleardevice(); rectangle(W\_xmin,W\_ymin,W\_xmax,W\_ymax); line(x,y,x1,y1);

line(0,0,600,0);

line(0,0,0,600);

if(y>W\_ymax) { rcode\_begin[0]=1; // Top

flag=1 ;

}

if(y<W\_ymin) {

rcode\_begin[1]=1; // Bottom flag=1;

}

if(x>W\_xmax) { rcode\_begin[2]=1; // Right flag=1;

}

if(x<W\_xmin) { rcode\_begin[3]=1; //Left flag=1;

}

//end point of Line if(y1>W\_ymax){ rcode\_end[0]=1; // Top flag=1;

}

if(y1<W\_ymin) {

rcode\_end[1]=1; // Bottom flag=1;

}

if(x1>W\_xmax){ rcode\_end[2]=1; // Right flag=1;

}

if(x1<W\_xmin){ rcode\_end[3]=1; //Left flag=1;

}

if(flag==0)

{

printf("No need of clipping as it is already in window");

}

flag=1; for(i=0;i<4;i++){

region\_code[i]= rcode\_begin[i] && rcode\_end[i] ; if(region\_code[i]==1)

flag=0;

}

if(flag==0)

{

printf("\n Line is completely outside the window");

}

else{

slope=(float)(y1-y)/(x1-x);

if(rcode\_begin[2]==0 && rcode\_begin[3]==1) //left

{

y=y+(float) (W\_xmin-x)\*slope ; x=W\_xmin;

}

if(rcode\_begin[2]==1 && rcode\_begin[3]==0) // right

{

y=y+(float) (W\_xmax-x)\*slope ; x=W\_xmax;

}

if(rcode\_begin[0]==1 && rcode\_begin[1]==0) // top

x=x+(float) (W\_ymax-y)/slope ; y=W\_ymax;

}

if(rcode\_begin[0]==0 && rcode\_begin[1]==1) // bottom

{

x=x+(float) (W\_ymin-y)/slope ; y=W\_ymin;

}

// end points

if(rcode\_end[2]==0 && rcode\_end[3]==1) //left

{

y1=y1+(float) (W\_xmin-x1)\*slope ; x1=W\_xmin;

}

if(rcode\_end[2]==1 && rcode\_end[3]==0) // right

{

y1=y1+(float) (W\_xmax-x1)\*slope ; x1=W\_xmax;

}

if(rcode\_end[0]==1 && rcode\_end[1]==0) // top

{

x1=x1+(float) (W\_ymax-y1)/slope ; y1=W\_ymax;

}

if(rcode\_end[0]==0 && rcode\_end[1]==1) // bottom

{

x1=x1+(float) (W\_ymin-y1)/slope ; y1=W\_ymin;

}

}

delay(1000); clearviewport();

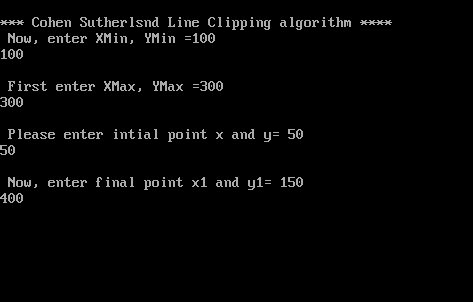
rectangle(W\_xmin,W\_ymin,W\_xmax,W\_ymax); line(0,0,600,0);

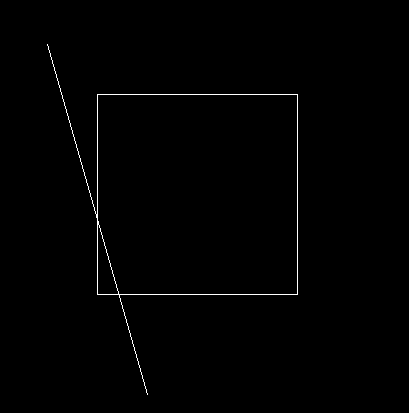
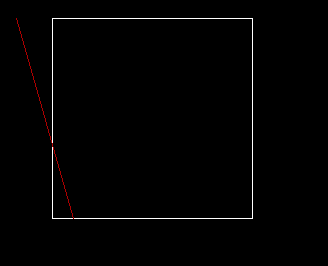
line(0,0,0,600);

setcolor(RED); line(x,y,x1,y1); getch(); closegraph();

}

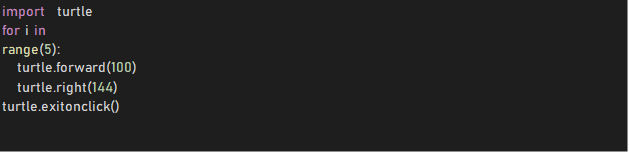
**OUTPUT:**

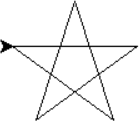


  
  
**PRACTICAL 11**

**Aim:-Implement the following shapes by using turtle Functions.**

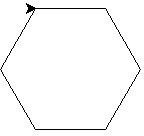
**Draw a star**





Star

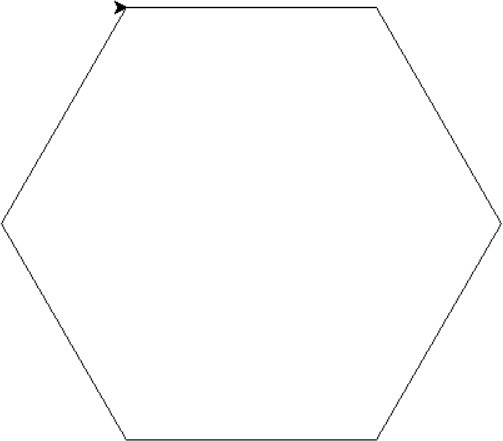
**Draw a Hexagon**



import turtle for i in range(6):

turtle.forward(200) turtle.right(60)

turtle.exitonclick()



#### **Draw a square inside another square box.**

from turtle import \*

x=200

for i in range(21): fd(x)

lt(90)

fd(x) x-=10

exitonclick()

