**EXPERIMENT -1**

**AIM : To Study Inbuilt Functions**

**1. Putpixel :**

**Description:**

Plots a pixel at a specified point in the specified color.

**Declaration:**

putpixel(int x,int y,int color)

**Program:-**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"c://turboc3//bgi");

putpixel(10,10,10);

getch();

}

**OUTPUT :**

****

2. **Line** :

**Description:**

Line function draw the line between two specified points. x1=start coordinate of x, x2=end coordinate of x,y1=start coordinate of y, y2=end coordinate of y.

**Declaration :** void line(int x1, int y1, int x2, int y2)

**Program:-**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm;

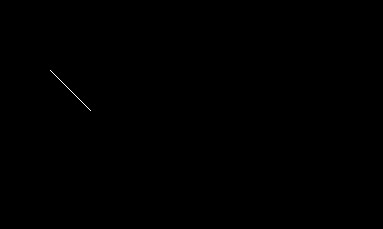
initgraph(&gd,&gm,"c://turboc3//bgi");

line(50,70,90,110);

getch();

}

**OUTPUT :**

****

3. **Setcolor :**

**Description:**

Sets the current drawing color. We can assign values from 0-15.

**Declaration :** void setcolor(int color)

**Program:-**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,”c:\\turboc3\\bgi”);

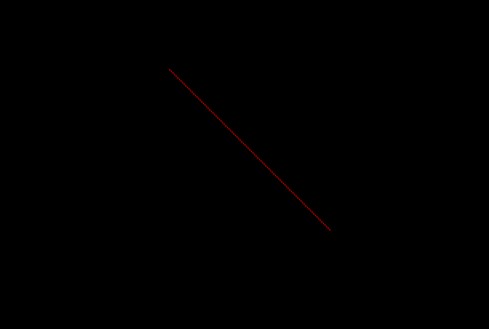
setcolor(4);

line(200,200,100,100);

getch();

}

**OUTPUT :**

****

4. **Circle :**

**Description:**

Centre and radius are specified. It helps to draw a circle. x= coordinate on x axis, y=coordinate on y axis, r= radius of the required circle.

**Declaration :** void setcolor(int color)

**Program:-**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm;

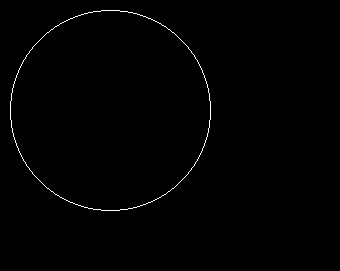
initgraph(&gd,&gm,"c:\\turboc3\\bgi");

circle(110,110,100);

getch();

}

**OUTPUT:**

****

5. **Arc** :

**Description:**

Draws a circular arc. 5 parameters are required x and y specifies coordinate on screen, stangle refers to start angle , endangle refers to end angle . x= coordinate of x, y=coordinate of y

stangle= start angle,endangle= end angle, r= radius of the arc

**Declaration :** void arc(int x, int y, int stangle, int endangle,int radius)

**Program:-**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm;

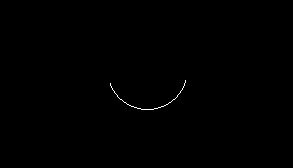
initgraph(&gd,&gm,"c://turboc3//bgi");

arc(300,300,200,345,40);

getch();

}

**OUTPUT:**

****

6. **Rectangle :**

**Description:**

Draws a rectangle in the cureent line style, thickness and drawing color. Left , top is the upper left corner of the rectangle, and (right bottom) is its lower right corner.

**Declaration :** void rectangle(int left, int top, int right, int bottom)

**Program:-**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm;

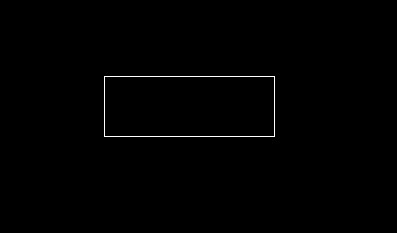
initgraph(&gd,&gm,"c://turboc3//bgi");

rectangle(180,210,350,150);

getch();

}

**OUTPUT:**

****

7. **Ellipse :**

**Decription:**

Draws an elliptical arc . Six parameters are specified. x and y are coordinates of center point . stangle refers to start angle and endangle refers to end angle . To draw an ellipse start angle = 0 and endangle=360 ideally.

**Declaration :** void ellipse(int x, int y, int stangle, int endangle,int xradius,int yradius) . x and y are coordinates of center point . stangle refers to start angle and endangle refers to end angle . To draw an ellipse start angle = 0 and endangle=360 ideally.

**Program:-**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm;

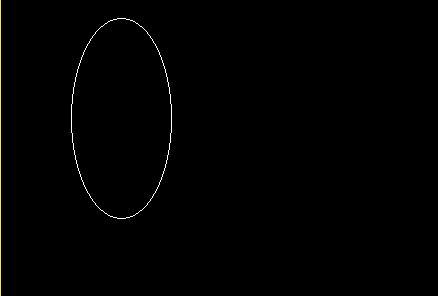
initgraph(&gd,&gm,"c://turboc3//bgi");

ellipse(120,120,0,360,50,100) ;

getch();

}

**OUTPUT:**

****

8. **Setfillstyle :**

**Decription:**

Sets the fill pattern and color. The enumerations fill patterns gives names for the predefined fill patterns

**Declaration:** void setfillstyle(FILL\_STYLE,COLOR);

**Fillelipse :**

Draws and fill an ellipse

**Declaration:** void fillellipse(int x,int y, int xradius,int yradius);

**Program:-**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"c://turboc3//bgi");

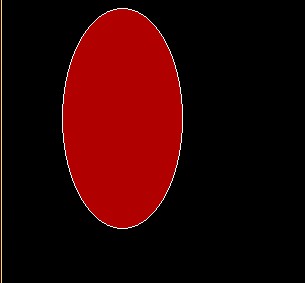
setfillstyle(SOLID\_FILL,RED);

fillellipse(120,120,60,110);

getch();

}

**OUTPUT:**

****

9. **Move to :**

**Description:**

Moves the current position CP to point specified (x,y)

**Line to:**

Draws a line from CP to specified point (x,y)

**Declaration :** void moveto(int x, int y)

void lineto(int x,int y)

**Program:-**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"c://turboc3//bgi");

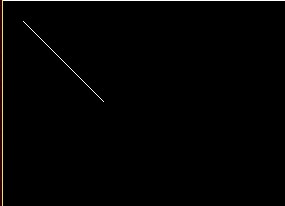
moveto(20,20);

lineto(100,100);

getch();

}

**OUTPUT:**

****

10**. Bar :**

**Description:**

A bar is a filled rectangular figure which appears on the screen. Right is the right hand coordinate of bar, left is the left hand coordinate of the bar, top is the upper coordinate of the bar, bottom is the lower coordinate of the bar.

**Declaration:**

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

**Program:**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm;

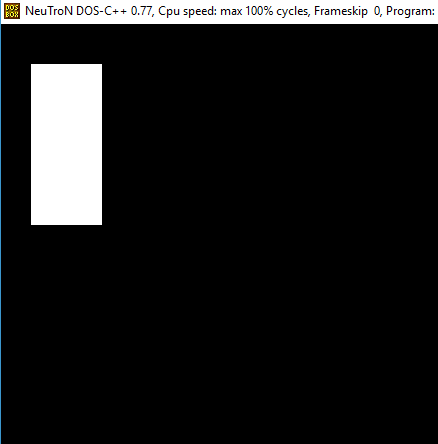
initgraph(&gd,&gm,"c://turboc3//bgi");

bar(100,200,30,40);

getch();

}

**Output:**



**EXPERIMENT - 2**

**AIM : Write a program to illustrate DDA Algorithm**

**DDA Algorithm:-**

Digital Differential Analyzer Algorithm is an incremental algorithm which works on slope intercept concept of line. For each increment in x axis by one unit, y is incremented by m units, where m is the slope of the line. Slope lies between -1 to 1

**Mathematical Calculation:**

**Case 1:** If m<=1

yi = mxi + c

yi+1=m ( xi + 1 ) + c

yi+1= mxi +m + c

yi+1=yi+m

yi+1-yi = m

Where m = dy/dx

**Case 2:** If m>1

yi = mxi + c

mxi+c+1=mxi+1 + c

mxi+1= mxi+1

xi+1= xi +1/m

xi+1 - xi =1/m

Where m = dy/dx

**Advantages:-**

1 Simplest algorithm for drawing the line

2 Faster than drawing the line with slope intercept equation because only increment is required .

**Disadvantages:-**

1 Insufficient as compared to other line drawing algorithm because we are not using exact value of pixel for drawing the line.

2 Everytime we are calling the floor function it makes DDA slower as compared to other line drawing algorithms.

**Program:-**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void DDA(int x1,int y1,int x2,int y2)

{

int dy=y2-y1;

int dx=x2-x1;

float m=dy/dx;

int x,y=y1;

for(x=x1;x<=x2;x++)

{

putpixel(x,y,10);

y=floor(y+m);

}

}

void main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\turboc3\\bgi");

DDA(100,100,200,200);

getch();

}

**OUTPUT :**

****

**EXPERIMENT – 3A**

**AIM : a) Write a program to illustrate MID Algorithm**

**MID Algorithm:-**

First find on what side of line M lies, if M is below the line NE pixel is selected else E pixel is selected. Slope of line lies between 0 and 1, increment is vertically or horizontally or both.

Check Condition if:

F(x,y) = 0 : x,y point lies on line

F(x,y) < 0 : x,y point lies below line

F(x,y) > 0 : x,y point lies above line

**Description:**

The midpoint line algorithm is due to Bresenham and was modified by Pitteway and Van Aken. It works as follows: Let the slope of the line be 0<=k<=1 . Suppose one approximate point P=(x,) is already determined. We have only two choices for the next point, namely E=((x+1),)  and NE=((x+1),(+1) ) and we should choose the one which is closer to k(x+1)+d. To determine the appropriate choice we proceed as follows:

* calculate the middle point M=(x+1,) .
* If the intersection point Q of the line with the vertical line connecting E and NE is below M, take E as next pixel.
* Otherwise take NE as next pixel.

**Mathematical Calculation:**

F(x,y)= ax+by+c = 0 … (1)

Also y = mx+B => y = x\*(dy/dx) + B => xdy –ydx +Bdx =0 … (2)

Comapring 1 and 2 , we get a=dy , b= -dx, c = Bdx

Mid Point is our decision parameter , so for initial mid point :-

F(x+1,y+1/2) = Dinitial = a(x+1) + b(y+(1/2)) + c … (3)

Where Dinitial = Initial mid point ( Decision Parameter)

If d<0 ( Mid point above line choosing E pixel)

New M:-

F(x+2,y+(1/2)) = a(x+2) + b(y+(1/2)) + c … (4)

Inc. E = 2\*eqn (4) – 2\*eqn(3) = 2a = 2dy [To avoid fractional] …(5)

If d>0 ( Mid point below line choosing NE pixel)

New M:-

F(x+2,y+(3/2)) = a(x+2) + b(y+(3/2)) + c … (6)

Inc. E = 2\*eqn (6) – 2\*eqn(3) = 2a+2b = 2dy-2dx [To avoid fractional] …(7)

Minitial => f(x+1,y+(1/2)) = a(x+1) + b(y+(1/2)) + c = ax + by + c + a + (b/2)

F(x,y)+a+(b/2) = 2a + b = 2dy – dx [To avoid fractional]

Using these results in algorithm

**Program**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void mid(int x1,int y1,int x2,int y2)

{

int dy=y2-y1;

int dx=x2-x1;

int m=2\*dy-dx;

int incrE=2\*dy;

int incrNE=2\*dy-2\*dx;

int x=x1;

int y=y1;

putpixel(x,y,7);

while(x<=x2)

{

if(m<=0)

{

m=m+incrE;

x++;

}

else if(m>0)

{

m=m+incrNE;

x++;

y++;

}

putpixel(x,y,7);

}

}

main()

{

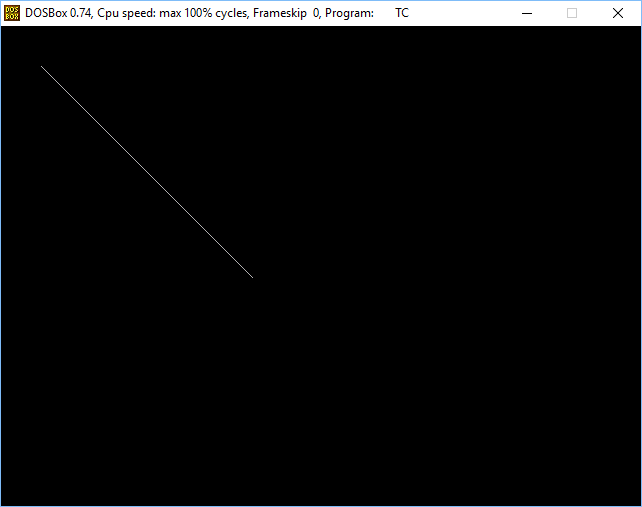
int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\turboc3\\bgi");

mid(40,40,250,250);

getch();

}



**EXPERIMENT – 3**B

**AIM : b) Write a program to illustrate Bresenham Algorithm**

**Bresenham Algorithm:-**

Check if line passing actually is closer to s or t, decision parameter d = d2-d1 is calculated , if d2-d1 < 0 t is selected else if d1-d2 > 0 : s is selected.

**Mathematical Calculation:**

y= m(xk + 1) + b

d1= y-yk = m(xk + 1) + b – yk

d2= (yk +1) – y = yk +1 - m(xk + 1) - b

d1-d2=2m(xk+1)-2yk+2b-1

A decision parameter pk for the kth step in the line algorithm can be obtained by simplifying above equation by substituting m = dy/dx.

pk=dx(d1-d2) = 2dy.xk - 2dx.yk + c

pk+1= 2dy.xk+1 - 2dx.yk+1 + c

p0=2dy-dx

**Program**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void brehsenham(int x1,int y1,int x2,int y2)

{

int dy=y2-y1;

int dx=x2-x1;

int d=2\*dy-dx;

int x=x1;

int y=y1;

int s=2\*dy;

int t=2\*dy-2\*dx;

putpixel(x,y,7);

while(x<=x2)

{

if(d<=0)

{

d=d+s;

x++;

}

else if(d>0)

{

x++;

y++;

d=d+t;

}

putpixel(x,y,7);

}

}

main()

{

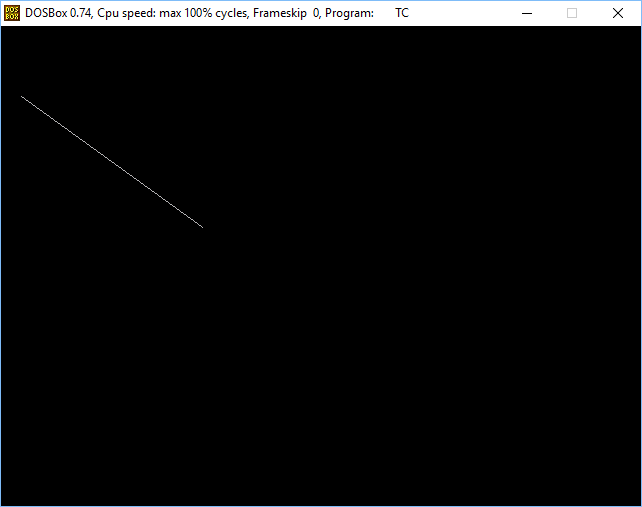
int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\turboc3\\bgi");

brehsenham(20,70,200,200);

getch();

}



**Experiment -4A**

**AIM: Write a program to illustrate Midpoint Circle Drawing Algorithm**

**Circle Drawing Algorithm:-**

A circle is a symmetrical fig. It has 8 way symmetry. Thus any circle generating algorithm take the advantage of circle symmetry to plot 8 points by calculating the coordinate of any 1 point.

Assuming we have just plotted the pixel at (*xk,yk*) so we need to choose between (*xk+*1*,yk*) and (*xk+*1*,yk-*1) . Our decision variable can be defined as:

If *pk* < 0 the midpoint is inside the circle and and the pixel at *yk* is closer to the circle

Otherwise the midpoint is outside and *yk-*1 is closer

To ensure things are as efficient as possible we can do all of our calculations incrementally

First consider:

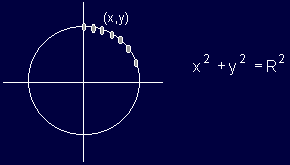
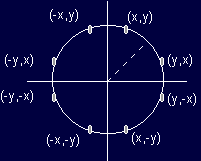
or:

where *yk+1* is either *yk* or *yk-*1 depending on the sign of *pk*

The first decision variable is given as:

Then if *pk* < 0 then the next decision variable is given as:

If *pk* > 0 then the decision variable is:

****

**Program:**

#include<stdio.h>

#include<conio.h>

#include<graphics.h> void circlem(int x1,int y1,int r)

{ int m=1-r; int incE=2\*x1+1; int incNE=2\*x1-2\*y1+1;

int x=0;

int y=r; while(x<=y)

{ if(m<=0)

{

m=m+incE; x++; } else

{

m=m+incNE; x++;

y--; }

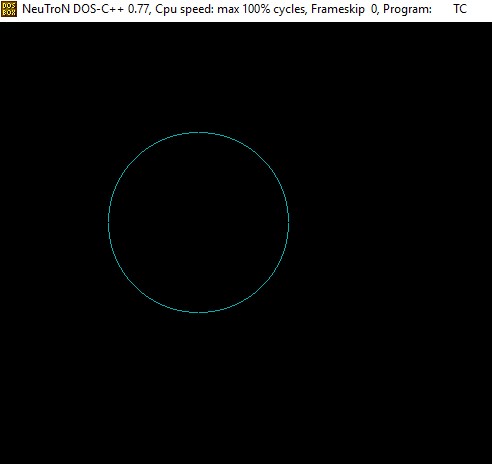
putpixel(x1+x,y1+y,3); putpixel(x1+x,y1-y,3); putpixel(x1-x,y1+y,3); putpixel(x1-x,y1-y,3); putpixel(x1+y,y1+x,3); putpixel(x1+y,y1-x,3); putpixel(x1-y,y1+x,3); putpixel(x1-y,y1-x,3); } } main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\turboc3\\bgi"); circle(200,200,90); getch(); }

**OUTPUT:-**



**Experiment -4B**

**AIM: Write a program to illustrate Bresenham Circle drawing Algorithm**

The objective of the algorithm is to find a path through the pixel grid using pixels which are as close as possible to solutions of x^2 + y^2 = r^2. At each step, the path is extended by choosing the adjacent pixel which satisfies x^2 + y^2 <= r^2 but maximizes x^2 + y^2 . Since the candidate pixels are adjacent, the arithmetic to calculate the latter expression is simplified, requiring only bit shifts and additions.

This algorithm starts with the circle equation. For simplicity, assume the center of the circle is at (0,0). We consider first only the first octant and draw a curve which starts at point(r,0) and proceeds counterclockwise, reaching the angle of 45.

The "fast" direction here (the basis vector with the greater increase in value) is the y direction. The algorithm always takes a step in the positive y direction (upwards), and occasionally takes a step in the "slow" direction (the negative x direction).

From the circle equation we obtain the transformed equation x^2 + y^2 - r^2 = 0, where r^2 is computed only a single time during initialization.

Let the points on the circle be a sequence of coordinates of the vector to the point (in the usual basis). Points are numbered according to the order in which they are drawn, with n=1 assigned to the first point (r,0).

For each point, the following holds:

\begin{align} x_n^2 + y_n^2 = r^2 \end{align}

This can be rearranged as follows:

\begin{align} x_n^2 = r^2 - y_n^2 \end{align}

And likewise for the next point:

\begin{align} x_{n+1}^2 = r^2 - y_{n+1}^2 \end{align}

In general, it is true that:

\begin{align} y_{n+1}^2 &= (y_n + 1)^2 \\ &= y_n^2 + 2y_n + 1 \end{align}

\begin{align} x_{n+1}^2 = r^2 - y_n^2 - 2y_n - 1 \end{align}

So we refashion our next-point-equation into a recursive one by substituting x_n^2 = r^2 - y_n^2:

\begin{align} x_{n+1}^2 = x_n^2 - 2y_n - 1 \end{align}

**Program:**

#include<stdio.h>

#include<conio.h> #include<graphics.h> void brescircle(int x1,int y1,int r)

{ int d=d-2\*r; int x=0; int y=r; while(x<=y)

{ if(d<=0)

{

d=d+4\*x+6; x++; y=y; } else if(d>0)

{ d=d+4\*(x-y)+10; x++;

y--; } putpixel(x1+x,y1+y,3); putpixel(x1+x,y1-y,3); putpixel(x1-x,y1+y,3); putpixel(x1-x,y1-y,3); putpixel(x1+y,y1+x,3); putpixel(x1+y,y1-x,3); putpixel(x1-y,y1+x,3); putpixel(x1-y,y1-x,3);

} } main()

{

int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\turboc3\\bgi"); brescircle(100,100,70); getch();

}

**Output:**



**Experiment -5**

**AIM: Write a program to illustrate Boundary Fill Algorithm**

**Boundary Fill Algorithm:-**

Boundary Fill is algorithm used for the purpose of coloring figures in computer graphics. It is so similar to Flood Fill that many are confused as to whether it is another variation of it. Here area gets colored with pixels of a chosen color as boundary this giving the technique its name. One can see the difference in the conditions that are there for planting the seeds. Boundary fill fills the chosen area with a color until the given colored boundary is found. This algorithm is also recursive in nature as the function returns when the pixel to be colored is the boundary color or is already the fill color.

If the boundary is specified in a single color, the fill algorithm proceeds outward pixel-by-pixel until the boundary color is encountered. This is called boundary fill algorithm

1.The boundary fill procedure accepts as input the coordinates of an interior point (x, y), a fill color, and a boundary color.  
2.Starting from (x, y), the procedure tests the neighboring positions to determine whether they are boundary color.  
3.If not, they are painted with the fill color, and the neighbors are tested.  
4.This process continues until all pixels up to the boundary color for the area have been tested Two methods  
 (i)4- connected:-4 neighbouring points are conected  
 (ii)8-connected:-correctly fill the interior of the area defined

**Program**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void boundary(int x,int y,int fc,int bc)

{

if(getpixel(x,y)!=fc&&getpixel(x,y)!=bc)

{

putpixel(x,y,5);

boundary(x+1,y,fc,bc);

boundary(x-1,y,fc,bc);

boundary(x,y+1,fc,bc);

boundary(x,y-1,fc,bc);

}

}

void main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\turboc3\\bgi");

setcolor(3);

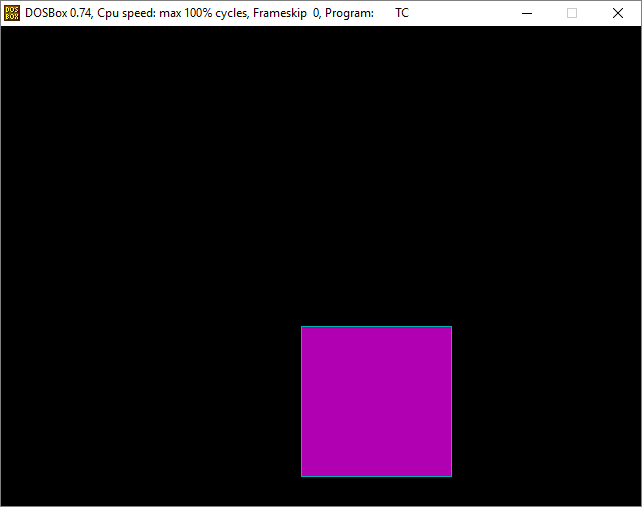
rectangle(100,100,150,150);

boundary(101,101,5,3);

getch();

}

**Output:**



**EXPERIMENT-6**

**Aim - Write a program to implement Flood Fill Algorithm for Polygon Filling.**

**Description**

Sometimes we come across an object where we want to fill the area and its boundary with different colors. We can paint such objects with a specified interior color instead of searching for particular boundary color as in boundary filling algorithm.

Instead of relying on the boundary of the object, it relies on the fill color. In other words, it replaces the interior color of the object with the fill color. When no more pixels of the original interior color exist, the algorithm is completed.

Once again, this algorithm relies on the Four-connect or Eight-connect method of filling in the pixels. But instead of looking for the boundary color, it is looking for all adjacent pixels that are a part of the interior.

**Algorithm**

void flood(int x,int y,int fillc)

{

int color=getpixel(x,y);

if(color!=fillc)

{

putpixel(x,y,fillc);

flood(x+1,y,fillc);

flood(x-1,y,fillc);

flood(x,y-1,fillc);

flood(x,y+1,fillc);

}

}

**Program**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

void flood(int x,int y,int fillc)

{

int color=getpixel(x,y);

if(color!=fillc)

{

putpixel(x,y,fillc);

flood(x+1,y,fillc);

// flood(x-1,y,fillc);

flood(x,y-1,fillc);

flood(x,y+1,fillc);

}

}

void main()

{

int gd=DETECT,gm;

clrscr();

initgraph(&gd,&gm,"C:\\TurboC3\\BGI");

setcolor(3);

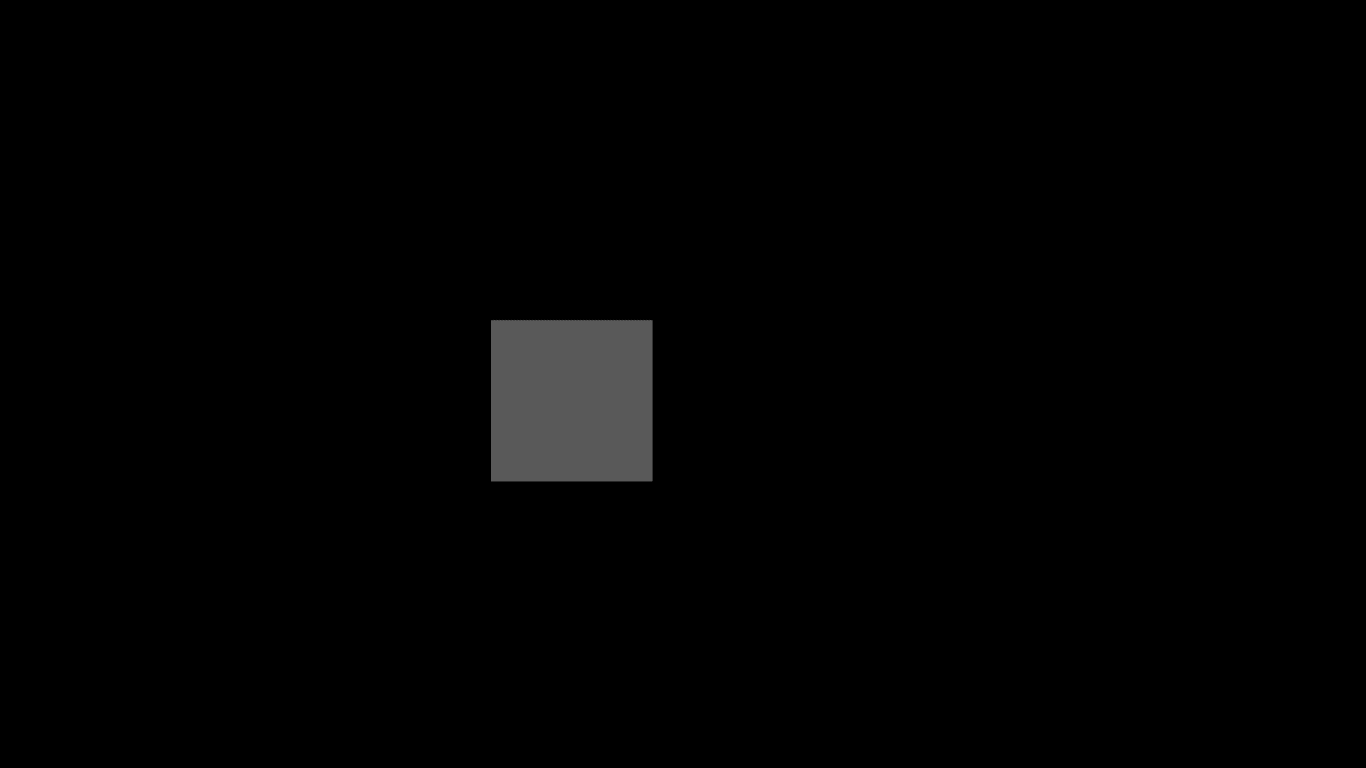
rectangle(200,200,300,300);

flood(201,201,3);

getch();

}

**Output**



**Experiment-7**

**Aim: WAP to perform Translation Transformation.**

**Description:**

A translation moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate (tx, ty) to the original coordinate (X, Y) to get the new coordinate (X’, Y’).



From the above figure, you can write that:

X’ = X + tx

Y’ = Y + ty

The pair (tx, ty) is called the translation vector or shift vector. The above equations can also be represented using the column vectors.

𝑃=[𝑋𝑌] 𝑃′=[𝑋′𝑌′] 𝑇=[𝑡𝑥𝑡𝑦]

We can write it as: P’ = P + T

**Source Code:**

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

int main()

{

int gd=DETECT, gm;

initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

line(100,100,150,150);

line(150,150,150,100);

line(100,100,150,100);

int Tx=100,Ty=100;

line(100+Tx,100+Ty,150+Tx,150+Ty);

line(150+Tx,150+Ty,150+Tx,100+Ty);

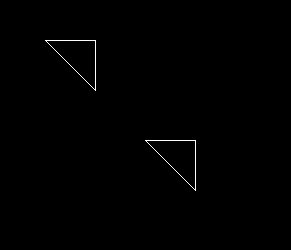
line(100+Tx,100+Ty,150+Tx,100+Ty);

getch();

return 0;

}

**Output:**



**Experiment-8**

**Aim: WAP to perform Scaling Transformation.**

**Description:**

To change the size of an object, scaling transformation is used. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result.

Let us assume that the original coordinates are (X, Y), the scaling factors are (SX, SY), and the produced coordinates are (X’, Y’). This can be mathematically represented as shown below:

X’ = X ∙ SX and Y’ = Y ∙ SY

The scaling factor SX, SY scales the object in X and Y direction respectively. The above equations can also be represented in matrix form as below:

X’ = X ∙ SX and Y’ = Y ∙ SY

The scaling factor SX, SY scales the object in X and Y direction respectively. The above equations can also be represented in matrix form as below:

[X’] = [SX 0] [X]

[y’] = [0 SY] [Y]

OR

P’ = P ∙ S

Where S is the scaling matrix. The scaling process is shown in the following figure.

****

**Figure: Before scaling process**

****

**Figure: After Scaling Process**

If we provide values less than 1 to the scaling factor S, then we can reduce the size of the object. If we provide values greater than 1, then we can increase the size of the object.

**Source Code:**

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

int main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

line(50,50,75,145);

line(75,145,100,120);

line(50,50,100,120);

int tx=2;

int ty=1;

line(50\*tx,50\*ty,75\*tx,145\*ty);

line(75\*tx,145\*ty,100\*tx,120\*ty);

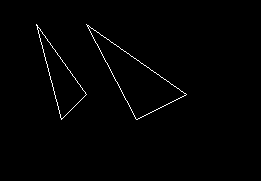
line(50\*tx,50\*ty,100\*tx,120\*ty);

getch();

return 0;

}

**Output:**

****

**Experiment-9**

**Aim: WAP to perform Reflection Transformation.**

**Description:**

Reflection is the mirror image of original object. In other words, we can say that it is a rotation operation with 180˚. In reflection transformation, the size of the object does not change.

The following figures show reflections with respect to X and Y axes, about the origin and about line y=x respectively.





**Source Code:**

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

int main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TurboC3\\BGI");

int xmax=getmaxx();

int ymax=getmaxy();

int xmid=xmax/2;

int ymid=ymax/2;

line(0,ymid,xmax,ymid);

line(xmid,0,xmid,ymax);

rectangle(xmid+20,ymid-20,xmid+60,ymid-60);

setcolor(2);

rectangle(xmid+20,ymid+20,xmid+60,ymid+60);

rectangle(xmid-20,ymid-20,xmid-60,ymid-60);

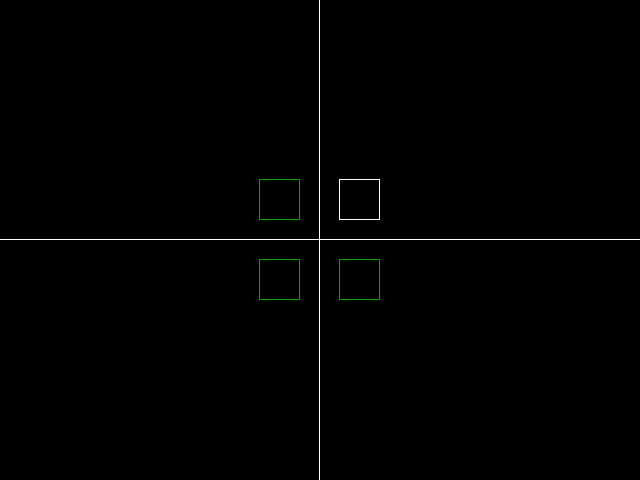
rectangle(xmid-20,ymid+20,xmid-60,ymid+60);

getch();

return 0;

}

**Output:**

****

**Experiment-10**

**Aim: WAP to perform Shear Transformation.**

**Description:**

A transformation that slants the shape of an object is called the shear transformation. There are two shear transformations **X-Shear** and **Y-Shear**. One shifts X coordinates values and other shifts Y coordinate values. However, in both the cases, only one coordinate changes its coordinates and other preserves its values. Shearing is also termed as **Skewing**.

**X-Shear**

The X-Shear preserves the Y coordinate and changes are made to X coordinates, which causes the vertical lines to tilt right or left as shown in below figure.

****

The X-Shear Transformation can be represented as:

X’ = X + Shx ∙ Y

Y’ = Y

**Y-Shear**

The Y-Shear preserves the X coordinates and changes the Y coordinates which causes the horizontal lines to transform into lines which slopes up or down as shown in the following figure.



The Y-Shear Transformation can be represented as:

Y’ = Y + Shy ∙ X

X’ = X

**Source Code:**

#include<iostream.h>

#include<conio.h>

#include<graphics.h>

int main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

rectangle(60,60,120,120);

int shx=2;

int shy=2;

setcolor(2);

rectangle(60,60+shy\*60,120,120+shy\*120);

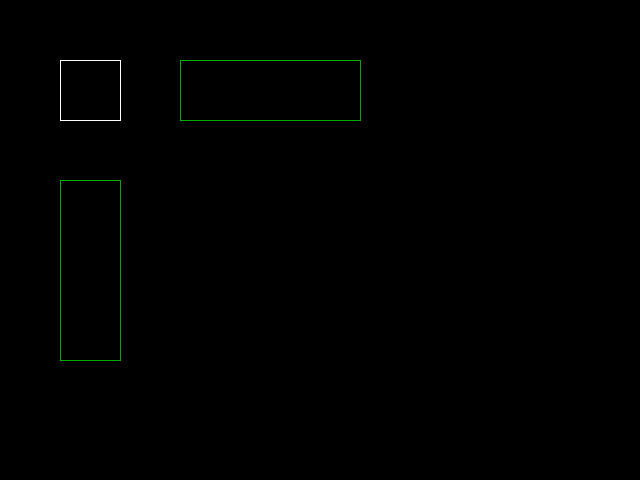
rectangle(60+shx\*60,60,120+shx\*120,120);

getch();

return 0;

}

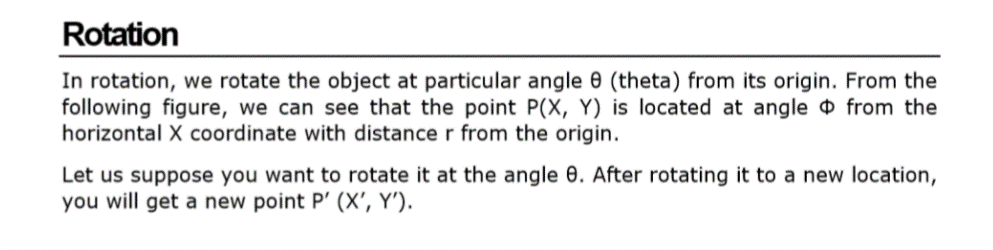
**Output:**

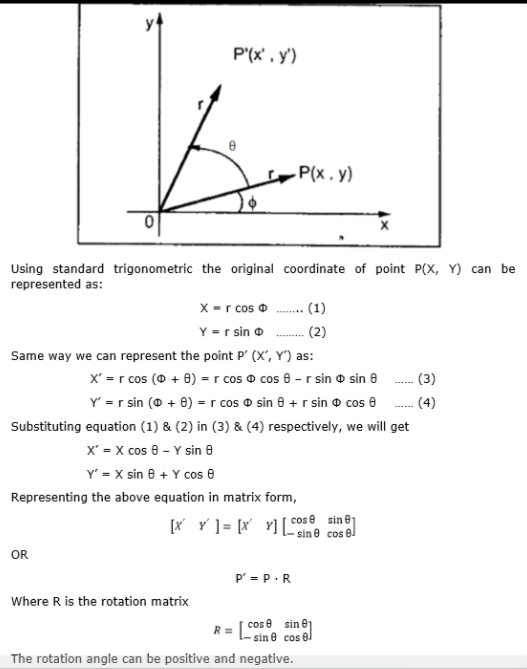
****

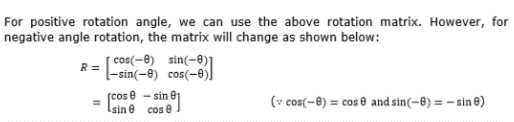
**Experiment-11**

**Aim: WAP to perform Rotation Transformation.**

**Description:**







Code :

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<math.h>

void main()

{

int gd=DETECT, gm;

initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

line(100,150,180,200);

float p;

printf("Enter angle by which rotation is to be performed :");

scanf("%f",&p);

p=p\*3.14/180;

int x1=100\*cos(p)-150\*sin(p);

int y1=100\*sin(p)+150\*cos(p);

int x2=180\*cos(p)-200\*sin(p);

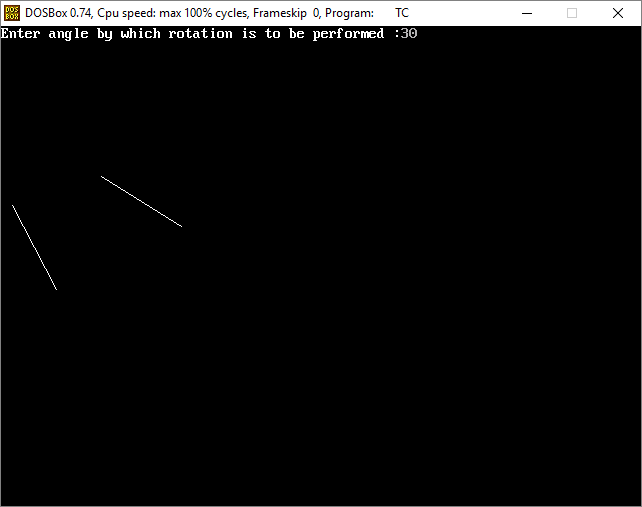
int y2=180\*sin(p)+200\*cos(p);

line(x1,y1,x2,y2);

getch();

}

Output :



**Experiment-12**

**Aim: WAP to make wallpaper using graphics function .**

**Code :**

#include<iostream>

#include<Windows.h>

using namespace std;

class naming

{

public:

void letter\_o();

void letter\_s();

void letter\_h();

void letter\_a();

void letter\_m();

void set\_posn(short,short);

void test\_m();

};

void naming::letter\_o()

{

int i, j, k, l = 0;

for (j = 7; j > 0; j--)

{

cout << " ";

}

cout << "\*";

cout << endl;

l++;

set\_posn(10,l);

for (i = 0; i < 3; i++)

{

for (k = 3; k > i; k--)

{

cout << " ";

}

cout << "\*";

for (int l = -3; l < (i + 2) \* 2; l++)

{

cout << " ";

}

for (k = 0; k < i; k++)

{

cout << " ";

}

cout << "\*";

cout << endl;

l++;

set\_posn(10,l);

if (i == 2)

{

set\_posn(11, l);

}

}

for (i = 3; i > 0; i--)

{

for (k = 2; k > i; k--)

{

cout << " ";

}

cout << "\*";

if (i == 3)

{

for (int l = ((i + 2) \* 3) + 1; l > 2; l--)

{

cout << " ";

}

}

else

{

for (int l = (i + 2) \* 3; l > 0; l--)

{

cout << " ";

}

}

cout << "\*";

cout << endl;

l++;

set\_posn(12,l);

}

for (j = 7; j > 0; j--)

{

cout << " ";

}

cout << "\*";

cout << endl;

l++;

set\_posn(12,l);

}

void naming::letter\_s()

{

int i, j;

for (i = 0; i < 5; i++)

{

cout << "\*";

}

cout << endl;

for (i = 0; i < 3; i++)

{

for (j = 0; j < 1; j++)

{

cout << "\*";

}

cout << endl;

}

for (i = 0; i < 5; i++)

{

cout << "\*";

}

cout << endl;

for (i = 0; i < 3; i++)

{

for (int l = 0; l < 4; l++)

{

cout << " ";

}

for (j = 0; j < 1; j++)

{

cout << "\*";

}

cout << endl;

}

for (i = 0; i < 5; i++)

{

cout << "\*";

}

cout << endl;

}

void naming::letter\_h()

{

int i, j, l,m=0;

for (i = 0; i < 8; i++)

{

if (i == 4)

{

for (i = 0; i < 5; i++)

{

cout << "\*";

}

cout << endl;

m++;

set\_posn(30, m);

}

for (j = 0; j < 1; j++)

{

cout << "\*";

}

for (l = 0; l < 3; l++)

{

cout << " ";

}

for (j = 0; j < 1; j++)

{

cout << "\*";

}

cout << endl;

m++;

set\_posn(30, m);

}

}

void naming::letter\_a()

{

int i, j, k,l=0;

for (i = 0; i < 8; i++)

{

if (i == 0)

{

for (k = 5; k > i; k--)

{

cout << " ";

}

for (j = 0; j < 1; j++)

{

cout << "\*";

}

}

else

{

for (k = 8; k > i; k--)

{

cout << " ";

}

for (j = 0; j < 1; j++)

{

cout << "\*";

}

for (int l = 1; l < i; l++)

{

cout << " ";

}

for (j = 0; j < 1; j++)

{

cout << "\*";

}

}

cout << endl;

if (i == 0)

{

l++;

set\_posn(39, l);

}

else

{

l++;

set\_posn(39, l);

}

}

}

void naming::letter\_m()

{

int i = 0, j, l, m = 0;

if (i == 0)

{

for (i = 0; i < 1; i++)

{

cout << "\*";

}

for (l = 0; l < 18; l++)

{

cout << " ";

}

for (i = 0; i < 1; i++)

{

cout << "\*";

}

cout << endl;

set\_posn(58, 1);

}

for (i = 0; i < 8; i++)

{

for (j = 0; j < 1; j++)

{

cout << "\*";

}

for (l = 0; l < i; l++)

{

cout << " ";

}

for (j = 0; j < 1; j++)

{

cout << "\*";

}

for (l = 8; l > i; l--)

{

cout << " ";

}

for (j = 0; j < 1; j++)

{

cout << "\*";

}

for (l = 0; l < i; l++)

{

cout << " ";

}

for (j = 0; j < 1; j++)

{

cout << "\*";

}

cout << endl;

m++;

set\_posn(58, m);

}

}

void naming::set\_posn(short a,short b)

{

COORD cord;

cord.X = a;

cord.Y = b;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), cord);

}

int main()

{

naming a;

HANDLE b = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(b, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

a.letter\_s();

SetConsoleTextAttribute(b, FOREGROUND\_RED | FOREGROUND\_INTENSITY);

Sleep(1000);

a.set\_posn(10,0);

a.letter\_o();

a.set\_posn(30, 0);

SetConsoleTextAttribute(b, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

Sleep(1000);

a.letter\_h();

a.set\_posn(42, 0);

SetConsoleTextAttribute(b, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

Sleep(1000);

a.letter\_a();

a.set\_posn(58, 0);

SetConsoleTextAttribute(b, FOREGROUND\_RED | FOREGROUND\_INTENSITY);

Sleep(1000);

a.letter\_m();

system("pause");

return 0;

}

/\*#include<iostream>

using namespace std;

class soham

{

public:

void star();

};

void soham::star()

{

int i, j,count=0;

for (i = 0; i < 5; i++)

{

for (j = 0; j <= i; j++)

{

cout << "\*";

}

cout << endl;

count++;

if (i == 4)

{

for (i = -10; i < count + 1; i++)

{

cout << "\b";

}

cout << "s";

}

}

}

int main()

{

soham a;

a.star();

system("pause");

return 0;

}\*/

//Setting Console Position

/\*#include<iostream>

#include<Windows.h>

using namespace std;

void test();

int main()

{

test();

system("pause");

return 0;

}

void test()

{

COORD cord;

cord.X = 1;

cord.Y = 1;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), cord);

}\*/

/\*#include<iostream>

#include<Windows.h>

using namespace std;

int main()

{

HANDLE a = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(a, FOREGROUND\_BLUE | FOREGROUND\_INTENSITY);

cout << "s" << endl;

Sleep(1000);

SetConsoleTextAttribute(a, FOREGROUND\_RED | FOREGROUND\_INTENSITY);

cout << "o" << endl;

system("pause");

return 0;

}\*/

/\*#include<iostream>

#include<windows.h>

using namespace std;

void s();

void h();

void e();

void r();

void y();

void l();

void gotoxy(int, int);

void main()

{

cout << "My name is:" << endl;

HANDLE H = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(H, FOREGROUND\_GREEN);

s();

SetConsoleTextAttribute(H, FOREGROUND\_BLUE);

h();

SetConsoleTextAttribute(H, FOREGROUND\_RED);

e();

SetConsoleTextAttribute(H, FOREGROUND\_GREEN);

r();

SetConsoleTextAttribute(H, FOREGROUND\_BLUE);

y();

SetConsoleTextAttribute(H, FOREGROUND\_RED);

l();

system("pause");

}

void s()

{

int i, j;

for (i = 0; i < 4; i++)

cout << "\* ";

for (i = 0; i < 3; i++)

cout << "\* " << endl;

for (i = 0; i <= 4; i++)

cout << "\* ";

cout << endl;

for (i = 0; i < 3; i++)

{

for (j = 0; j <= 3; j++)

{

cout << " ";

}

cout << "\* \n";

}

for (i = 0; i <= 4; i++)

cout << "\* ";

}

void h(){

int i, j, n = 1, m = 11;

for (i = 0; i < 3; i++)

{

gotoxy(11, n);

cout << "\* ";

for (j = 0; j < 6; j++)

{

cout << " ";

}

cout << "\*\n";

n++;

}

for (j = 0; j <= 4; j++)

{

gotoxy(m, 4);

cout << "\* ";

m = m + 2;;

}

cout << endl;

n = 5;

for (i = 5; i <= 8; i++)

{

gotoxy(11, n);

cout << "\* ";

for (j = 0; j < 6; j++)

{

cout << " ";

}

cout << "\*" << endl;

n++;

}

}

void e()

{

int i, m = 22, n = 1;

for (i = 0; i < 4; i++)

{

gotoxy(m, 1);

cout << "\* ";

m = m + 2;

}

for (i = 0; i < 4; i++)

{

gotoxy(22, n);

cout << "\*" << endl;

n++;

}m = 22;

for (i = 0; i < 4; i++)

{

gotoxy(m, 4);

cout << "\* ";

m = m + 2;

}n = 5;

for (i = 0; i < 4; i++)

{

gotoxy(22, n);

cout << "\*" << endl;

n++;

}m = 22;

for (i = 0; i < 4; i++)

{

gotoxy(m, 8);

cout << "\* ";

m = m + 2;

}

}

void r()

{

int i, j, m = 31, n = 2;

for (i = 0; i <= 4; i++)

{

gotoxy(m, 1);

cout << "\* ";

m = m + 2;

}

cout << endl;

for (i = 0; i < 2; i++)

{

gotoxy(31, n);

cout << "\* ";

for (j = 0; j < 6; j++)

{

cout << " ";

}

cout << "\*\n";

n++;

}m = 31;

for (i = 0; i <= 4; i++)

{

gotoxy(m, 4);

cout << "\* ";

m = m + 2;

}

cout << endl;

for (i = 0; i <= 8; i++)

{

gotoxy(31, n);

cout << "\* ";

for (j = 0; j <= i; j++)

{

cout << " ";

}

if (i = j)

cout << "\* \n";

n++;

}

}

void y(){

int i, j, k = 2, n = 1;

for (i = 0; i < 3; i++)

{

gotoxy(43, n);

for (j = 0; j < i; j++)

{

cout << " ";

}

cout << "\*";

for (j = 0; j <= k; j++)

{

cout << " ";

}

cout << "\*";

k--;

cout << endl;

n++;

}

for (i = 0; i <= 4; i++)

{

gotoxy(43, n);

for (j = 0; j < 3; j++)

{

cout << " ";

}

cout << " \*\n";

n++;

}

}

void l()

{

int i, n = 1, m = 55;

for (i = 0; i < 7; i++)

{

gotoxy(55, n);

cout << "\*\n";

n++;

}

for (i = 0; i <= 4; i++)

{

gotoxy(m, 8);

cout << "\* ";

m = m + 2;

}

}

void gotoxy(int x, int y)

{

HANDLE H;

COORD pos;

pos.X = x;

pos.Y = y;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), pos);

}\*/



**EXPERIMENT-13**

**Aim: Write a program to Zoom In an object.**

**Source Code:**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<dos.h>

void main()

{

intgd=DETECT,gm;

int top=20,left=20, right=120, bottom=120,midx,midy;

initgraph(&gd,&gm,"C:\\TurboC3\\BGI");

for(right=120,bottom=120;right<=400 && bottom<=400;right+=20,bottom+=20);

{

clearviewport();

setviewport(top,left,bottom,right,1);

midx = (left+right)/2;

midy = (top+bottom)/2;

circle(midx,midy,midy-left-20);

setfillstyle(2,3);

floodfill(midx,midy,3);

delay(1000);

}

getch();

}

**EXPERIMENT-14**

**Aim : Write a program to reverse zoom an object.**

**Source Code:**

#include<stdio.h>

#include<conio.h>

#include<graphics.h>

#include<dos.h>

void main()

{

intgd=DETECT,gm;

int top=20,left=20, right=400, bottom=400,midx,midy;

initgraph(&gd,&gm,"C:\\TurboC3\\BGI");

for(right=400,bottom=400;right>=120 && bottom>=120;right-=20,bottom-=20);

{

clearviewport();

setviewport(top,left,bottom,right,1);

midx = (left+right)/2;

midy = (top+bottom)/2;

circle(midx,midy,midy-left-20);

delay(1000);

}

getch();

}