OBJECTIVES:

# LAB 1

To implement DDA Algorithm for drawing a line segment between two given end points A(x0,y0) and B(x1,y1).

## THEORY:

DDA algorithm is an incremental scan conversion method. Here we perform calculations at each step using the results from the preceding step. The characteristic of the DDA algorithm is to take unit steps along one coordinate and compute the corresponding values along the other coordinate. The unit steps are always along the coordinate of greatest change, e.g. if dx = 10 and dy = 5, then we would take unit steps along x and compute the steps along y. In DDA we need to consider two cases; One is slope of the line less than or

equal to one (|m| ≤1) and slope of the line greater than one (m| > 1).

1. When |m| ≤ 1 means y1-y0 = x1-x0 or y1-y0< x1-x0 and therefore we assume x to be the major axis. Here we sample x axis at unit intervals and find the y values corresponding to each x value. We have the slope equation as

∆ y = m ∆ x

y0-y1 = m (x1-x0)

so, xk+1= xk+1 and yk+1= yk+m

1. When |m| > 1 means y2-y1 > x2-x1 and therefore we assume y to be the major axis. Here we sample y axis at unit intervals and find the x values corresponding to each y value. We have the slope equation as

∆ y = m ∆ x

y2-y1 = m (x2-x1)

so, yk+1= yk+1 and xk+1= xk+1/m

## ALGORITHM:

Step1: Start

Step2: Read two end points P1(x1, y1) and P2 (x2,y2)

Step3: Calculate: dx = x2 – x1

dy = y2 – y1 Step4: calculate: slope m = dy/dx Step5: if abs(m)<1

y = y1

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

plot the pixel position with specified color: setPixel(x, round(y)) else if abs(m>1)

for (y=y1; y<=y2; y++) x = x+1/m

plot the pixel position with specified color: setPixel(round(x), y) else

for(x=x1;x<=x2;x++); x = x+1;

y = y+1;

plot the pixel position with specified color: putpixel(round(x),round(y)) Step6: Close the graph and Stop.

## PROGRAM:

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

#include <math.h> #include <dos.h> void main( )

{

float x,y,x1,y1,x2,y2,dx,dy,step; int i,gd=DETECT,gm;

initgraph(&gd,&gm,"c:\\turboc3\\bgi"); printf("Enter the value of x1 and y1 : "); scanf("%f%f",&x1,&y1);

printf("Enter the value of x2 and y2: "); scanf("%f%f",&x2,&y2);

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

else step=dy; dx=dx/step; dy=dy/step; x=x1;

y=y1; i=1;

while(i<=step)

{

putpixel(x,y,5); x=x+dx; y=y+dy;

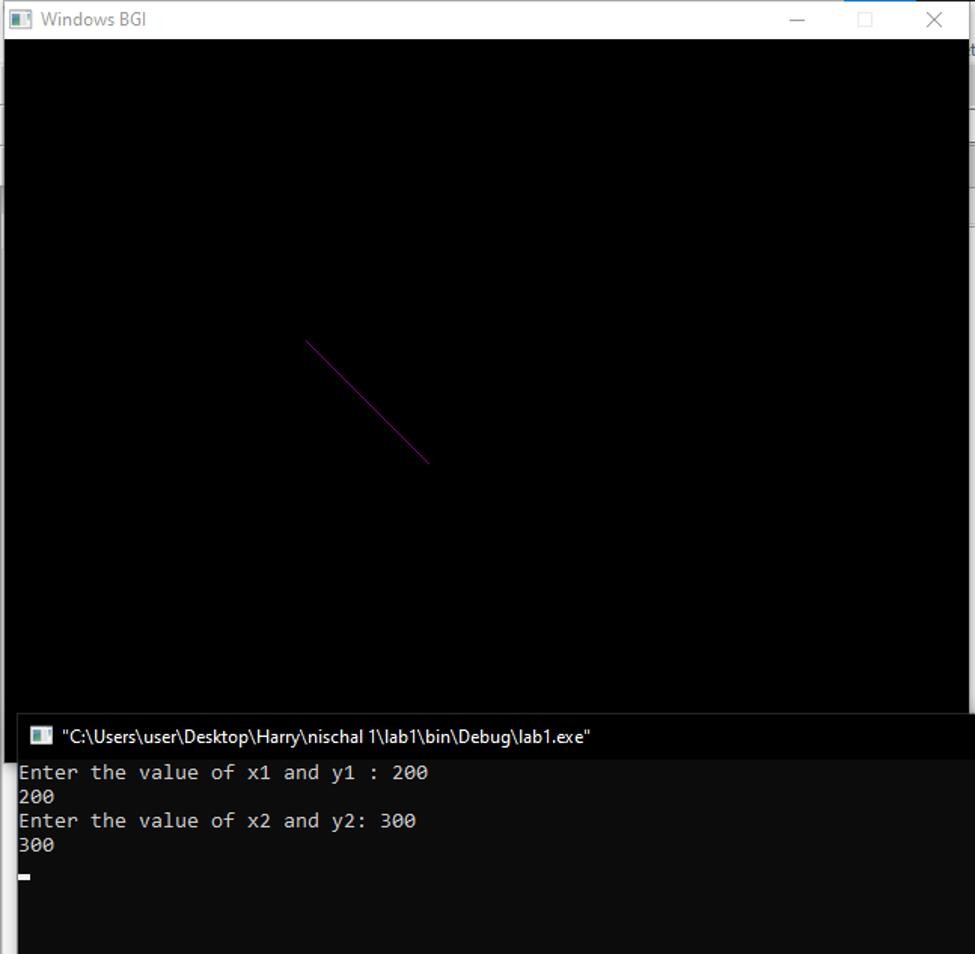
i=i+1; delay(100);

}

closegraph();

}

## OUTPUT:



OBJECTIVE:

# LAB 2

To implement BLA Algorithm for drawing a line segment between two given end points A (x1, y1) and B(x2, y2).

## THEORY:

This algorithm is used for scan converting a line. It was developed by Bresenham. It is an efficient method because it involves only integer addition, subtractions, and multiplication operations. These operations can be performed very rapidly so lines can be generated quickly.

In this method, next pixel selected is that one who has the least distance from true line. The method works as follows:

Assume a pixel P1'(x1',y1'),then select subsequent pixels as we work our may to the night, one pixel position at a time in the horizontal direction toward P2'(x2',y2').

The next pixel is

1. Either the one to its right (lower-bound for the line)
2. One top its right and up (upper-bound for the line)

## ALGORITHM:

Step 1 : Start

Step 2: Input starting point P1(x1, y1) and ending pointP2 (x2,y2)

Step 3 : Calculate the slope(m) of the required Line. Step 4 : Identify the value of slope(m). m = dy/dx

Step 4.1 : If slope(m) is Less than 1 i.e: m < 1

* Step 4.1.1 : Calculate the constants dx, dy,, and (2dy – 2dx) and get the first value for the decision parameter as -
* p0 = 2dy − dx
* Step 4.1.2 : At each Xk along the line, starting at k = 0, perform the following test −
* If pk < 0, the next point to plot is (xk + 1,yk) and pk+1 = pk + 2dy else
* plot (xk+1,yk + 1)
* pk+1 = pk + 2dy − 2dx
* Step 4.1.3 : Repeat until x2>=x1..
  + Step 4.2 : If slope(m) is greater than or equal to 1 i.e: m >= 1
* Step 4.2.1 : Calculate the constants dx, dy, and (2dx – 2dy) and get the first value for the decision parameter as -
* p0 = 2dx − dy
* Step 4.2.2 : At each yk along the line, starting at k = 0, perform the following test −
* If pk < 0, the next point to plot is (xk,yk + 1) and pk+1 = pk + 2dx else
* plot (xk + 1,yk+1)
* pk+1 = pk + 2dx − 2dy
* Step 4.2.3 : Repeat until y2>=y1. Step 5 : Exit.

## PROGRAM:

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

void drawline(int x0, int y0, int x1, int y1)

{

int dx, dy, p, x, y; dx=x1-x0;

dy=y1-y0; x=x0; y=y0; p=2\*dy-dx;

while(x<x1)

{

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;

}

}

int main()

{

int gdriver=DETECT, gmode, error, x0, y0, x1, y1; initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");

printf("Enter co-ordinates of first point: "); scanf("%d%d", &x0, &y0);

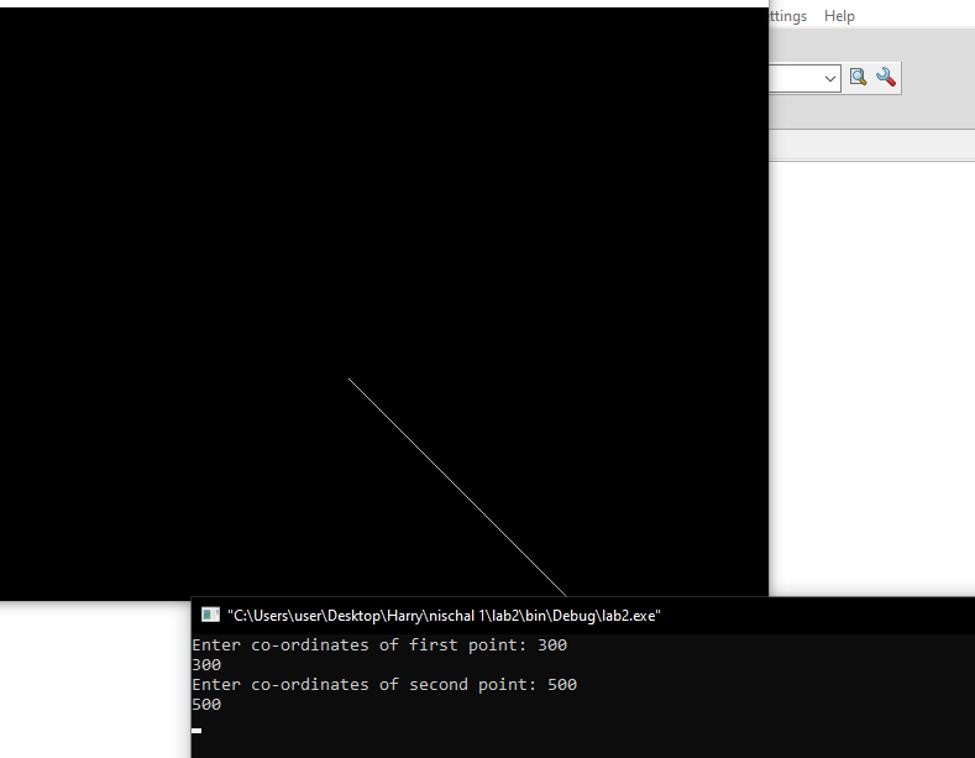
printf("Enter co-ordinates of second point: "); scanf("%d%d", &x1, &y1);

drawline(x0, y0, x1, y1); getch();

return 0;

}

OUTPUT:



# LAB 3

## OBJECTIVE:

To implement midpoint circle generation algorithm or Bresenham’s circle algorithm

for drawing a circle of given center (x,y) and radius r.

## THEORY:

Circle have the property of being highly symmetrical, which is handy when it comes to drawing them of a display screen.

* 1. We know that there are 360 degrees in a circle. First we see that a circle is symmetrical about the x axis, so only the first 180 degrees need to be calculated.
  2. Next we see that its also symmetrical about the y axis, so now we only need to calculate the first 90 degrees.
  3. Finally we see that the circle is also symmetrical about the 45 degree diagonal axis, so we only need to calculate the first 45 degrees.
  4. We only need to calculate the values on the boarder of the circle in the first octant. The other values may be determined symmetry.

Bresenham’s circle algorithm calculates the locations of the pixels in the first 45 degrees. It assumes that the circle is centered on the origin. So for every pixel (x,y) it calculates, we draw a pixel in each of the eight octants of the circle. This is done till when the value of the y coordinate equals the x coordinate. The pixel positions for determining symmetry are given in the below algorithm.

## ALGORITHM:

1. Input radius r and circle center (xc,yc), then set the coordinates for the first point on the circumference of a circle centered on the origin as:

(xc,yc) = (0,r)

1. Calculate the initial value of the decision parameter as:

P0 = 5/4 − 1

1. Starting with k = 0 at each position xk, perform the following test. If pk <0, the next point along the circle centered on (0,0) is (xk+1,yk) and:

Pk+1 = Pk+ 2\*xk+1+1

Otherwise the next point along the circle (xk+1,yk-1) and:

Pk+1 = Pk+ 2\*xk+1+1 – 2\*yk+1

1. Determine symmetry points in the other seven octants
2. Move each calculated pixel position (x,y) onto the circular path centered (xc,yc) to plot the coordinate values:
3. Repeat steps 3 to 5 until x>=y.

## PROGRAM:

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

void drawCircle(int xc, int yc, int x, int y)

{

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

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

}

void circleBres(int xc, int yc, int r)

{

int x = 0, y = r; int d = 3 - 2 \* r;

drawCircle(xc, yc, x, y); while (y >= x)

{

x++;

if (d > 0)

{

y--;

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

}

else

d = d + 4 \* x + 6; drawCircle(xc, yc, x, y); delay(50);

}

}

int main()

{

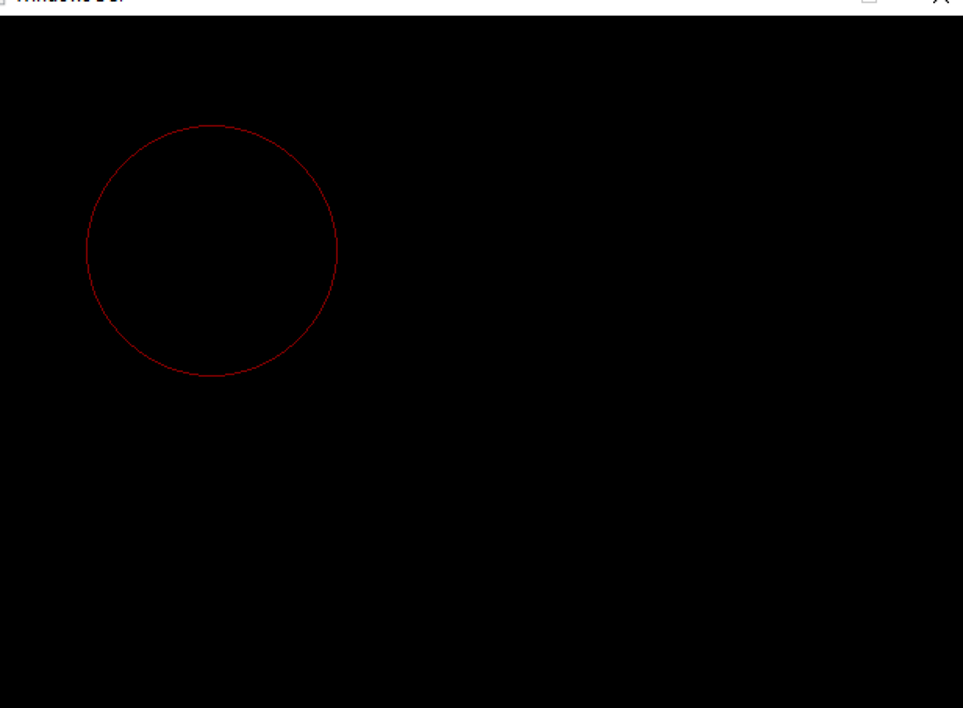
int xc = 150, yc = 150, r = 80; int gd = DETECT, gm; initgraph(&gd, &gm, "");

circleBres(xc, yc, r); getch();

return 0;

}

## OUTPUT:



OBJECTIVES:

# LAB 4

To implement midpoint circle generation algorithm or Bresenham’s circle algorithm

for drawing an ellipse of given center (x,y) and major, minor axis rx, ry.

## THEROY:

In Ellipse, Symmetry between quadrants exists Not symmetric between the two octants of a quadrant Thus, we must calculate pixel positions along the elliptical arc through one quadrant and then we obtain positions in the remaining 3 quadrants by symmetry. The next pixel is chosen based on the decision parameter.

## ALGORITHM:

1. Take input radius along x axis and y axis and obtain center of ellipse.
2. Initially, we assume ellipse to be centered at origin and the first point as : (x, y0)= (0, ry).
3. Obtain the initial decision parameter for region 1 as: p10=ry ^2 +1/4rx ^2 -rx ^2 ry
4. For every xk position in region 1 :

If p1k<0 then the next point along the is (xk+1 , yk) and p1k+1=p1k+2ry ^2 xk+1+ry ^2 Else, the next point is (xk+1, yk-1 )

And p1k+1=p1k+2ry ^2 xk+1 – 2rx ^2 yk+1+ry ^2

1. Obtain the initial value in region 2 using the last point (x0, y0) of region 1 as: p20=ry ^2 (x0+1/2)^2 +rx ^2 (y0-1)^2 - rx ^2 ry ^2 .
2. At each yk in region 2 starting at k =0 perform the following task.

If p2k>0 the next point is (xk, yk-1) and p2k+1=p2k-2rx ^2 yk+1+rx ^2

1. Else, the next point is (xk+1, yk -1) and p2k+1=p2k+2ry ^2 xk+1 -2rx ^2 yk+1+rx ^2
2. Now obtain the symmetric points in the three quadrants and plot the coordinate value as: x=x+xc, y=y+yc
3. Repeat the steps for region 1 until 2ry ^2 x>=2rx ^2 y.

## PROGRAM:

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

void ellipse(int xc,int yc,int rx,int ry)

{

int gm=DETECT,gd; int x, y, p;

initgraph(&gm,&gd,"C:\\TC\\BGI"); x=0;

y=ry;

p=(ry\*ry)-(rx\*rx\*ry)+((rx\*rx)/4); while((2\*x\*ry\*ry)<(2\*y\*rx\*rx))

{

putpixel(xc+x,yc-y,RED); putpixel(xc-x,yc+y,RED); putpixel(xc+x,yc+y,RED); putpixel(xc-x,yc-y,RED); if(p<0)

{

x=x+1; p=p+(2\*ry\*ry\*x)+(ry\*ry);

}

else

{

x=x+1;

y=y-1;

p=p+(2\*ry\*ry\*x+ry\*ry)-(2\*rx\*rx\*y);

}

}

p=((float)x+0.5)\*((float)x+0.5)\*ry\*ry+(y-1)\*(y-1)\*rx\*rx-rx\*rx\*ry\*ry; while(y>=0)

{

putpixel(xc+x,yc-y,RED); putpixel(xc-x,yc+y,RED); putpixel(xc+x,yc+y,RED); putpixel(xc-x,yc-y,RED); if(p>0)

{

y=y-1;

p=p-(2\*rx\*rx\*y)+(rx\*rx);

}

else

{

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

p=p+(2\*ry\*ry\*x)-(2\*rx\*rx\*y)-(rx\*rx);

}

}

getch(); closegraph();

}

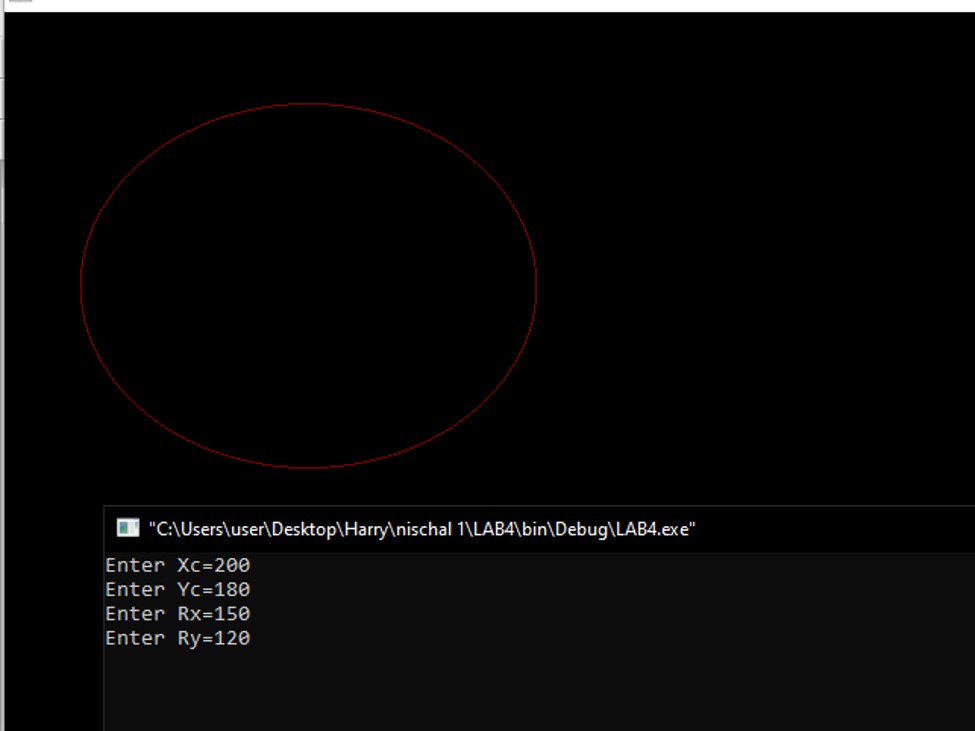
int main()

{

int xc,yc,rx,ry; printf("Enter Xc="); scanf("%d",&xc); printf("Enter Yc="); scanf("%d",&yc); printf("Enter Rx="); scanf("%d",&rx); printf("Enter Ry="); scanf("%d",&ry); ellipse(xc,yc,rx,ry); getch();

}

## OUTPUT:



OBJECTIVE:

# LAB 5

To apply the basic 2D transformations such as translation, scaling, rotation, shearing and reflection for a given 2D object.

## THEORY:

We have to perform 2D transformations on 2D objects. Here we perform transformation on a line segment. The 2D transformations are:

1. Translation

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′

1. Scaling

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.

1. Rotation

In rotation, we rotate the object at particular angle θ theta from its origin. From the following figure, we can see that the point P (X,Y) is located at angle φ from the horizontal X coordinate with distance r from the origin. Let us suppose you want to rotate it at the angle θ. After rotating it to a new location, you will get a new point P’ (X′,Y′).

1. Reflection

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.

1. Shearing

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.

## ALGORITHM:

1. Start
2. Initialize the graphics mode.
3. Construct a 2D object (use Drawpoly()) e.g. (x,y)
4. A) Translation
   1. Get the translation value tx, ty
   2. Move the 2d object with tx, ty (x’=x+tx,y’=y+ty)
   3. Plot (x’,y’)
5. Scaling
   1. Get the scaling value Sx,Sy
   2. Resize the object with Sx,Sy (x’=x\*Sx,y’=y\*Sy)
   3. Plot (x’,y’)
6. Rotation
   1. Get the Rotation angle
   2. Rotate the object by the angle ф x’=x cos ф - y sin ф

y’=x sin ф - y cosф

* 1. Plot (x’,y’)

1. Shearing
   1. Get shearing parameters Shx , Shy
   2. Shearing in X axis Xnew = Xold + Shx x Yold Ynew = Yold

Shearing in Y axis Xnew = Xold

Ynew = Yold + Shy x Xold

## PROGRAM:

#include<iostream> #include<graphics.h> #include<math.h> using namespace std; int main()

{

int gd=DETECT,gm,s; initgraph(&gd,&gm,(char\*)"");

cout<<"1.Translation\n2.Rotation\n3.Scaling\n4.Reflection\n5.Shearing "<<endl; cout<<"Selection:";

cin>>s; switch(s)

{

case 1:

{ int x1=250,y1=200,x2=350,y2=250;

int tx=60,ty=60;

cout<<"Rectangle before translation"<<endl; setcolor(3);

rectangle(x1,y1,x2,y2); setcolor(4);

cout<<"Rectangle after translation"<<endl; rectangle(x1+tx,y1+ty,x2+tx,y2+ty); getch();

break;

}

case 2:

{ long x1=250,y1=200,x2=350,y2=250;

double a;

cout<<"Rectangle with rotation"<<endl; setcolor(3);

rectangle(x1,y1,x2,y2); cout<<"Angle of rotation:"; cin>>a;

a=(a\*3.14)/180;

long xr=x1+((x2-x1)\*cos(a)-(y2-y1)\*sin(a));

long yr=y1+((x2-x1)\*sin(a)+(y2-y1)\*cos(a)); setcolor(2);

rectangle(x1,y1,xr,yr); getch();

break;

}

case 3:

{

int x1=50,y1=50,x2=75,y2=75,y=4,x=4;

cout<<"Before scaling"<<endl; setcolor(3); rectangle(x1,y1,x2,y2); cout<<"After scaling"<<endl; setcolor(10); rectangle(x1\*x,y1\*y,x2\*x,y2\*y); getch();

break;

}

case 4:

{

int x1=250,y1=350,x2=600,y2=350,x3=350,y3=400;

cout<<"triangle before reflection"<<endl; setcolor(3);

line(x1,y1,x2,y2);

line(x1,y1,x3,y3);

line(x2,y2,x3,y3);

cout<<"triangle after reflection"<<endl; setcolor(5);

line(x1,-y1+500,x2,-y2+500); line(x1,-y1+500,x3,-y3+500); line(x2,-y2+500,x3,-y3+500);

getch(); break;

}

case 5:

{

int x1=400,y1=100,x2=600,y2=100,x3=400,y3=200,x4=600,y4=200,shx=2;

cout<<"Before shearing of rectangle"<<endl; setcolor(3);

line(x1,y1,x2,y2);

line(x1,y1,x3,y3);

line(x3,y3,x4,y4);

line(x2,y2,x4,y4);

cout<<"After shearing of rectangle"<<endl; x1=x1+shx\*y1;

x2=x2+shx\*y2; x3=x3+shx\*y3; x4=x4+shx\*y4; setcolor(13); line(x1,y1,x2,y2);

line(x1,y1,x3,y3);

line(x3,y3,x4,y4);

line(x2,y2,x4,y4); getch();

}

default:

{

cout<<"Invalid Selection"<<endl; break;

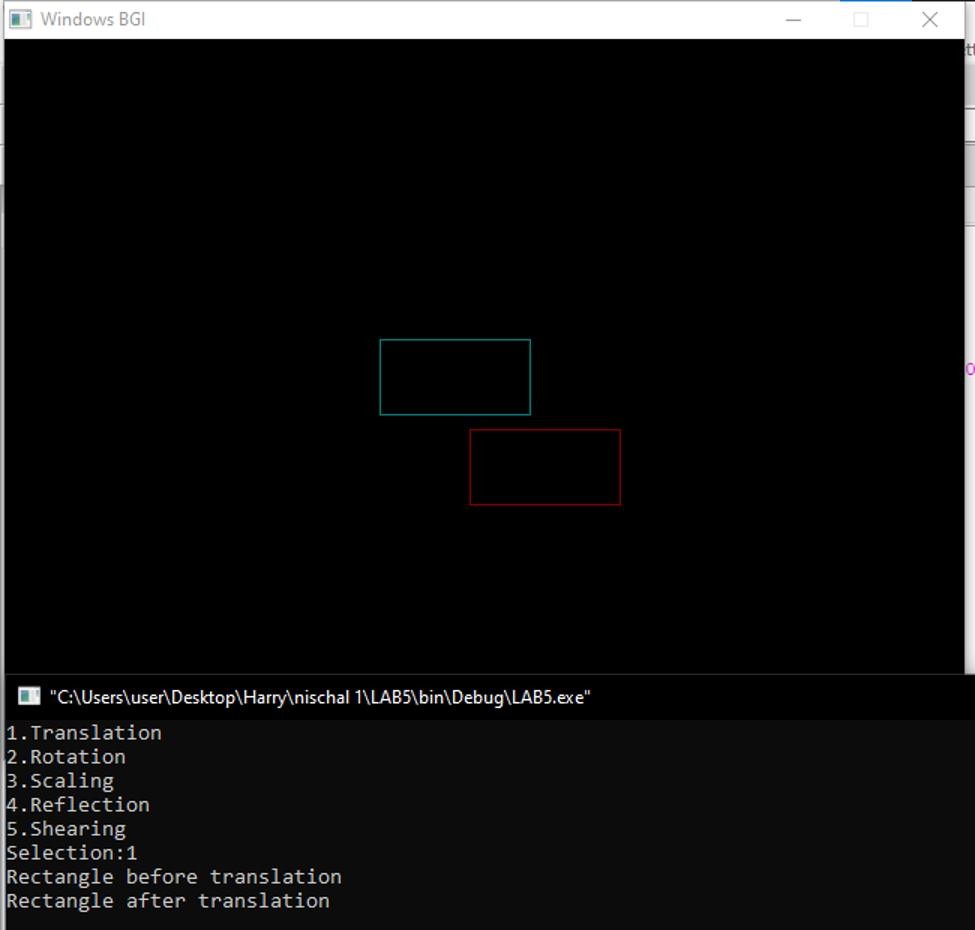
}

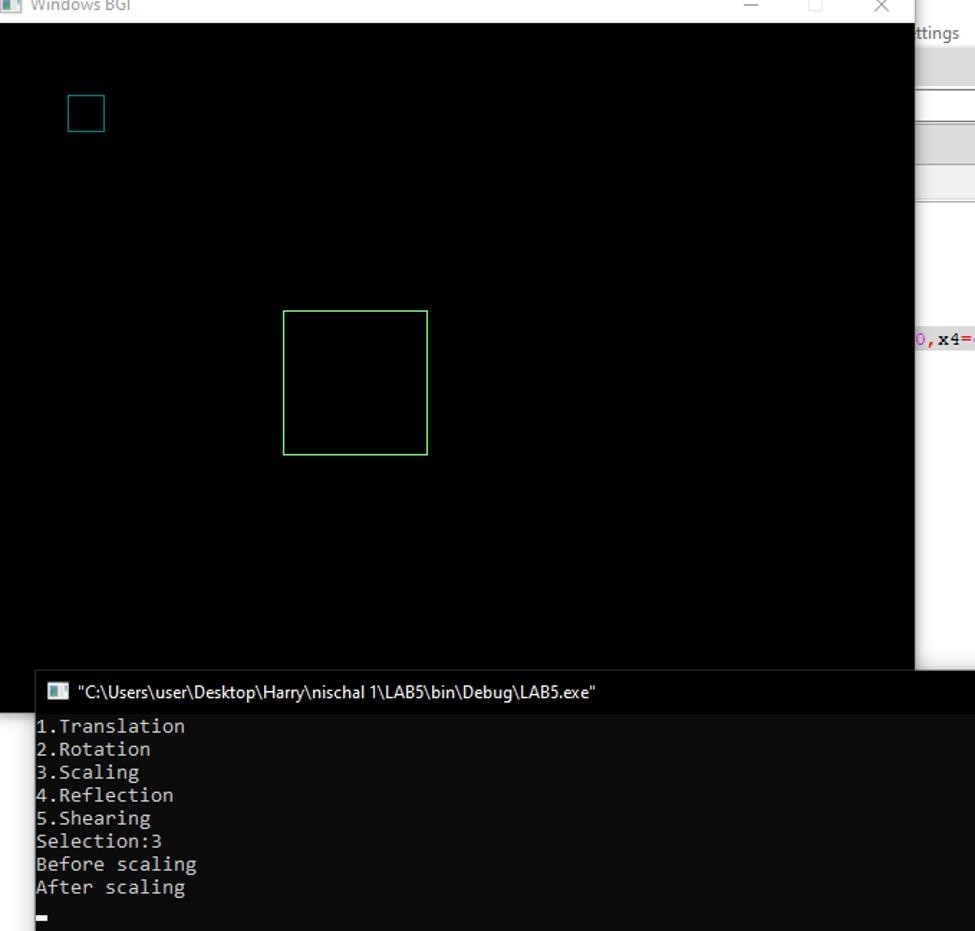
}

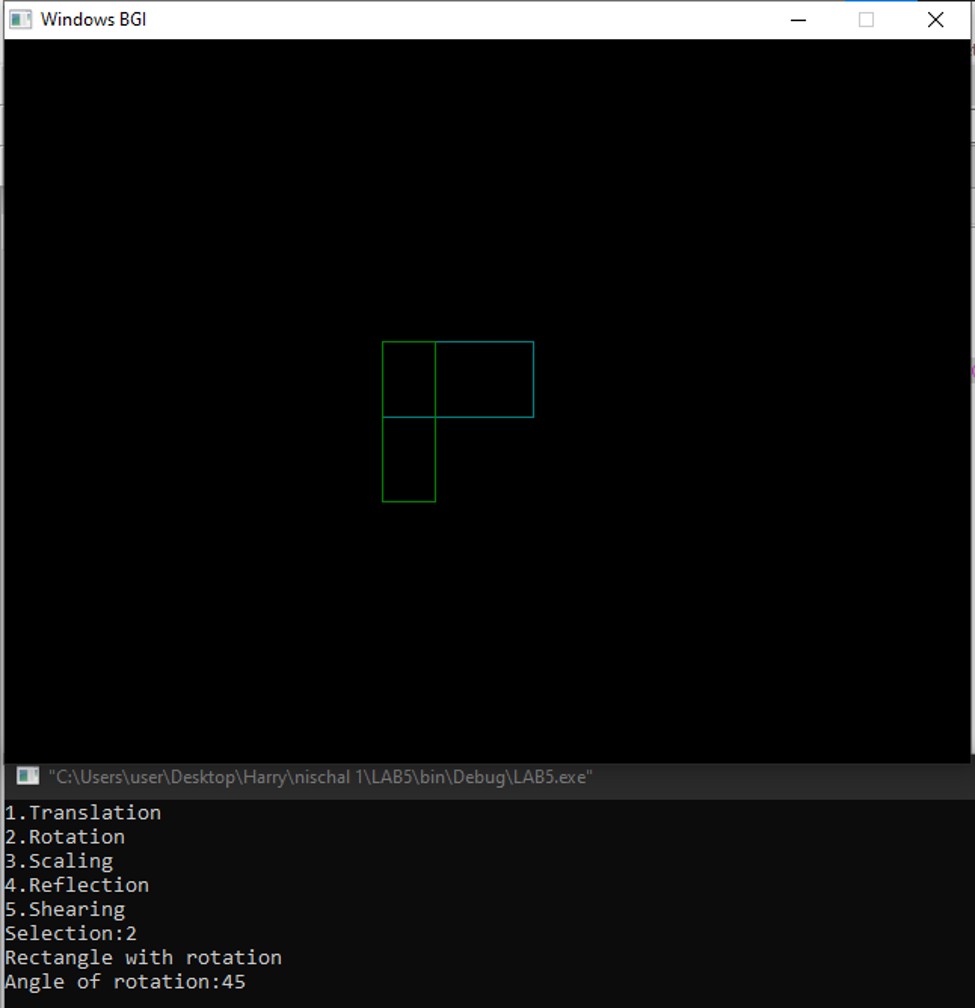
closegraph(); return 0;

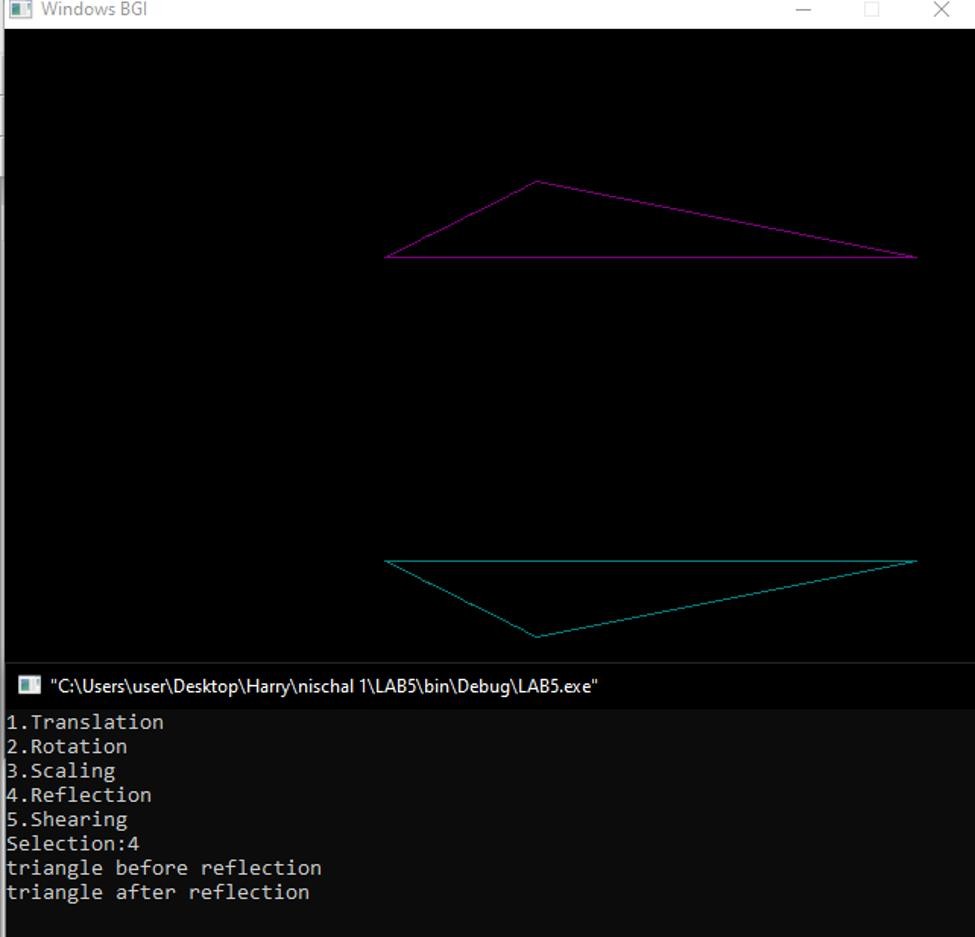
}

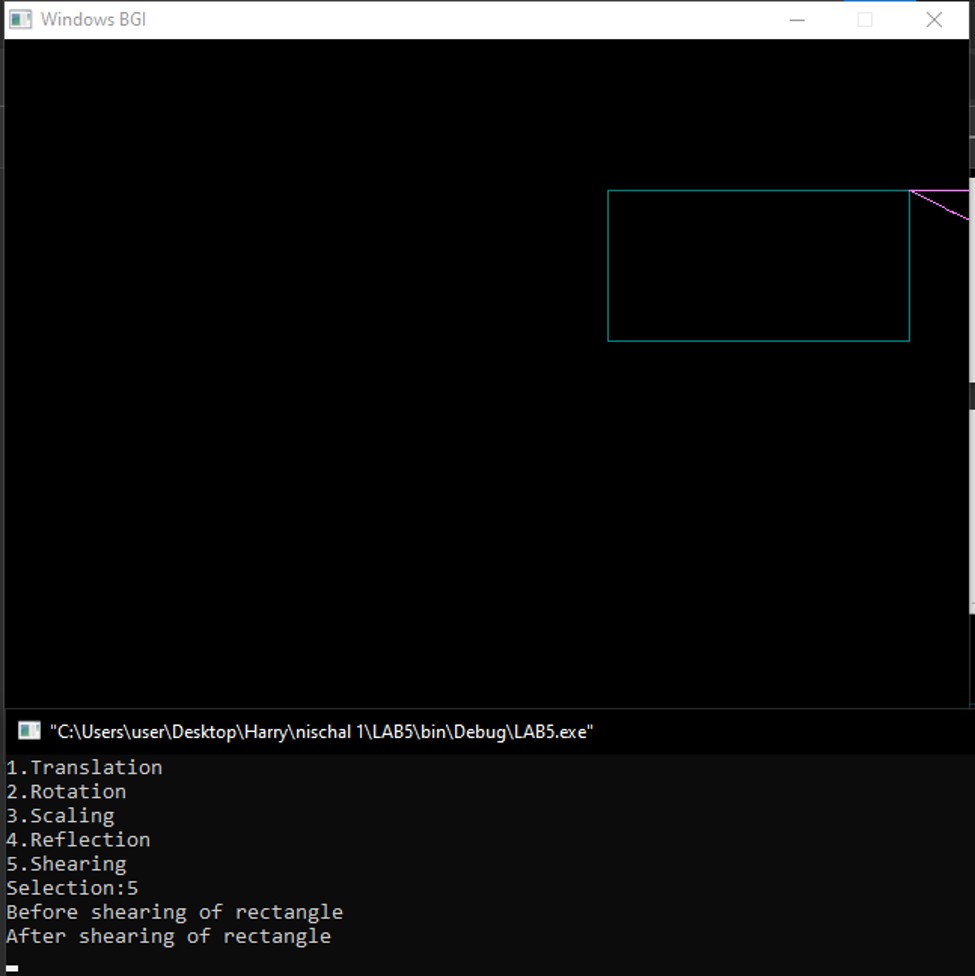
## OUTPUT:











OBJECTIVE:

# LAB 6

To implement the basic 2D flood filling in 2D.

## THEORY:

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

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

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

## ALGORITHM:

1. Procedure floodfill (x, y,fill\_ color, old\_color: integer)
2. If (getpixel (x, y)=old\_color)

{

1. setpixel (x, y, fill\_color);
2. fill (x+1, y, fill\_color, old\_color);
3. fill (x-1, y, fill\_color, old\_color);
4. fill (x, y+1, fill\_color, old\_color);
5. fill (x, y-1, fill\_color, old\_color);

}

}

## PROGRAM:

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

int floodfill(int x,int y,int old,int newcol)

{

int current; current=getpixel(x,y); if(current==old)

{

delay(5); putpixel(x,y,newcol); floodfill(x+1,y,old,newcol); floodfill(x-1,y,old,newcol); floodfill(x,y+1,old,newcol); floodfill(x,y-1,old,newcol); floodfill(x+1,y+1,old,newcol); floodfill(x-1,y+1,old,newcol); floodfill(x+1,y-1,old,newcol); floodfill(x-1,y-1,old,newcol);

}

}

int main()

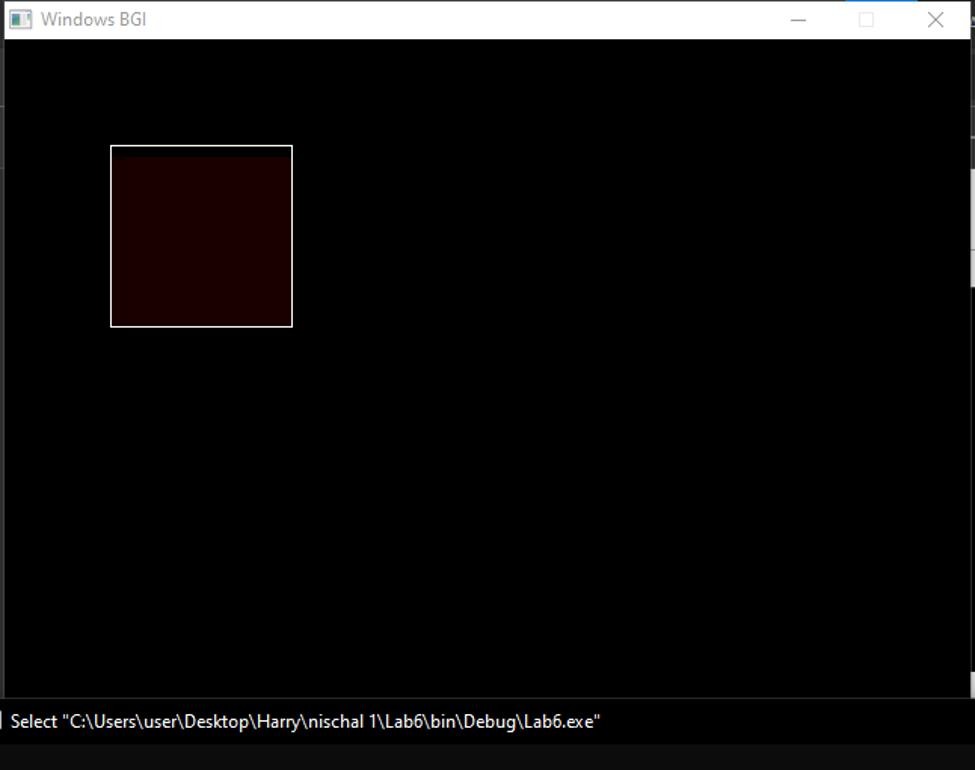
{

int gd=DETECT,gm; initgraph(&gd,&gm,"C:\\TURBOC3\\BGI"); rectangle(70,70,190,190); floodfill(80,80,0,25);

getch(); closegraph();

}

## OUTPUT:



OBJECTIVE:

# LAB 7

To implement the Scan line algorithm and seed fill algorithm in 2D.

## THEORY:

Two basic approaches are followed in area filling on raster systems. In the first approach overlap intervals for scan lines that cross the area are determined per scan line. Remember that a scan line is a horizontal line of pixels that can be plotted on the display device when the electron beam traverses the display area horizontally during one horizontal retrace. Second approach begins with an interior point and fills the area moving outward from this point until the boundary condition is reached. An algorithm following the first approach is classified as scan line algorithm and that falling under second class is called seed fill algorithm. Simple objects such as polygons, circles etc. are efficiently filled with scan line fill algorithms and more complex regions use the seed fill method. The scan line algorithm is mostly used in general graphics packages.

## ALGORITHM:

1. A seed pixel on a scan line within the area is popped from a stack containing the seed pixel.
2. The line or span containing the seed pixel is filled to the right & left of the seed pixel including the seed pixel itself until a boundary is found.
3. The extreme left and extreme right unprocessed pixel in the span are saved as X- left and X-right respectively.
4. The scan line above and below the current scan lines are examined in the range X- left to X-right for any contiguous span of either boundary pixels or previously filled pixels. If any such span is found it is simply crossed over. The extreme right pixel in all the unfilled spans on these scan lines within the same range is marked as a seed pixel and pushed onto the stack.

## PROGRAM:

#include <conio.h> #include <iostream> #include <graphics.h> #include <stdlib.h> using namespace std; class point

{

public:

int x,y;

};

class poly

{

private:

point p[20];

int inter[20],x,y;

int v,xmin,ymin,xmax,ymax; public:

int c;

void read(); void calcs(); void display(); void ints(float); void sort(int);

};

void poly::read()

{

int i;

cout<<"\n Scan Fill Algorithm ";

cout<<"\n Enter Number Of Vertices Of Polygon: "; cin>>v;

if(v>2)

{

for(i=0;i<v; i++) //ACCEPT THE VERTICES

{

cout<<"\nEnter co-ordinate no. "<<i+1<<" : "; cout<<"\n\tx"<<(i+1)<<"=";

cin>>p[i].x; cout<<"\n\ty"<<(i+1)<<"="; cin>>p[i].y;

}

p[i].x=p[0].x;

p[i].y=p[0].y; xmin=xmax=p[0].x; ymin=ymax=p[0].y;

}

else

cout<<"\n Enter valid no. of vertices.";

}

void poly::calcs()

{

for(int i=0;i<v;i++)

{

if(xmin>p[i].x)

xmin=p[i].x;

if(xmax<p[i].x)

xmax=p[i].x;

if(ymin>p[i].y)

ymin=p[i].y;

if(ymax<p[i].y)

ymax=p[i].y;

}

}

void poly::display()

{

int ch1; char ch='y'; float s,s2; do

{

cout<<"\n\nMENU:"; cout<<"\n\n\t1 . Scan line Fill "; cout<<"\n\n\t2 . Exit "; cout<<"\n\nEnter your choice:"; cin>>ch1;

switch(ch1)

{

case 1: s=ymin+0.01; delay(100); cleardevice(); while(s<=ymax)

{

ints(s);

sort(s); s++;

}

break; case 2:

exit(0);

}

cout<<"Do you want to continue?: "; cin>>ch;

}while(ch=='y' || ch=='Y');

}

void poly::ints(float z)

{

int x1,x2,y1,y2,temp; c=0;

for(int i=0;i<v;i++)

{

x1=p[i].x;

y1=p[i].y;

x2=p[i+1].x;

y2=p[i+1].y; if(y2<y1)

{

temp=x1; x1=x2;

x2=temp; temp=y1; y1=y2; y2=temp;

}

if(z<=y2&&z>=y1)

{

if((y1-y2)==0)

x=x1; else

{

x=((x2-x1)\*(z-y1))/(y2-y1); x=x+x1;

}

if(x<=xmax && x>=xmin) inter[c++]=x;

}

}

}

void poly::sort(int z) // sorting

{

int temp,j,i; for(i=0;i<v;i++)

{

line(p[i].x,p[i].y,p[i+1].x,p[i+1].y);

}

delay(100);

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

{

delay(100); line(inter[i],z,inter[i+1],z);

}

}

int main() //main

{

int cl; initwindow(500,600); cleardevice();

poly x; x.read();

x.calcs(); cleardevice();

cout<<"\n\tEnter The Color You Want :(In Range 0 To 15 )->"; //selecting color cin>>cl;

setcolor(cl); x.display();

closegraph(); //closing graph getch();

return 0;

}

## OUTPUT:

