# Q1. DDA Line Drawing algorithm

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

void main()

{

int gd=DETECT,gm ;

int x1,x2,y1,y2,i,step,dx,dy,xn,yn ;

initgraph(&gd,&gm,"C:\\TURBOC3\\BGI"); printf("Enter the 1st point co-ordinates :"); scanf("%d%d",&x1,&y1);

printf("Enter the 2nd point co-ordinates :"); scanf("%d%d",&x2,&y2);

cleardevice();

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

if(abs(dx)>abs(dy)) step=abs(dx);

else step=abs(dy);

xn=dx/step; yn=dy/step;

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

{

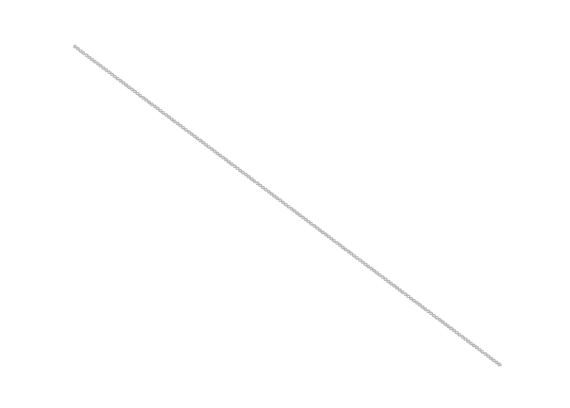
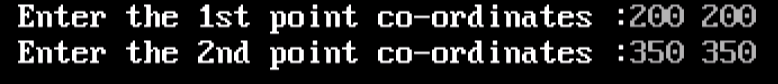
putpixel(x1,y1,WHITE); delay(50);

x1=x1+xn; y1=y1+yn;

}

getch();

}



Q2. Bresenham’s Line Drawing #include<stdio.h> #include<graphics.h>

#include<conio.h>

void main() {

int gd = DETECT, gm;

int x1, y1, x2, y2, dx, dy, x, y, pk;

// Initialize graphics mode

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

// Input the start and end points

printf("Enter the 1st point coordinates (x1, y1): "); scanf("%d%d", &x1, &y1);

printf("Enter the 2nd point coordinates (x2, y2): "); scanf("%d%d", &x2, &y2);

// Calculate differences dx = x2 - x1;

dy = y2 - y1;

// Determine the initial decision parameter pk = (2 \* dy) - dx;

// Set starting point x = x1;

y = y1;

// Draw the initial pixel

putpixel(x, y, WHITE);

// Bresenham's Line Algorithm for positive slope while (x < x2) {

x=x+1;

if (pk >= 0) { y=y+1;

pk += (2 \* dy) - (2 \* dx);

} else {

pk += 2 \* dy;

}

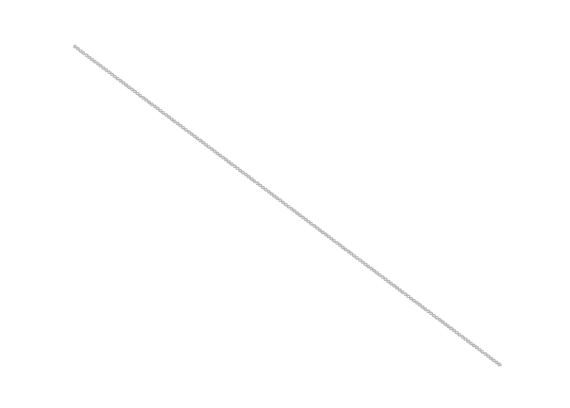
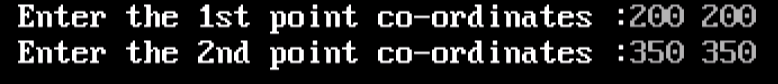
putpixel(x, y, WHITE);

delay(10); // Add delay for visualization

}

getch();

}



# Q3. Bresenham's Circle Drawing Algorithm

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

void main()

{

int gd=DETECT,gm ;

int xc=300,yc=300,x,y,r,d; initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");

printf("Enter radius of circle : "); scanf("%d",&r);

x=0;

y=r;

// d is decision parameter d=(3-(2\*r));

while(x<=y)

{

x=x+1; if(d<0)

d=d+(4\*x)+6; else

{

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

y=y-1;

}

delay(50);

putpixel(xc+x,yc+y,10); putpixel(xc+y,yc+x,10); putpixel(xc-y,yc+x,10); putpixel(xc-x,yc+y,10); putpixel(xc+y,yc-x,10); putpixel(xc+x,yc-y,10); putpixel(xc-y,yc-x,10);

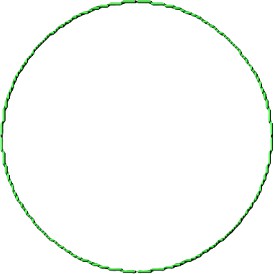
putpixel(xc-x,yc-y,10);

}

getch();

}





# Q4. Mid-Point Line Drawing Algorithm

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

void main()

{

int gd=DETECT,gm ;

// pk is decision parameter int x1,y1,x2,y2,dx,dy,x,y,p ;

initgraph(&gd,&gm,"C:\\TURBOC3\\BGI"); printf("Enter the 1st point co-ordinates :"); scanf("%d%d",&x1,&y1);

printf("Enter the last point co-ordinates :"); scanf("%d%d",&x2,&y2);

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

p=((2\*dy)-dx); // calculate p x=x1;

y=y1;

while(x1<=x2) // checking for 2 condition of p

{

delay(50); if(p>=0)

{

putpixel(x,y,10); p=p+dy-dx; y=y+1;

}

else

{

putpixel(x,y,10); p=p+dy;

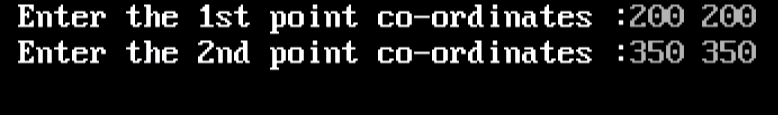
}

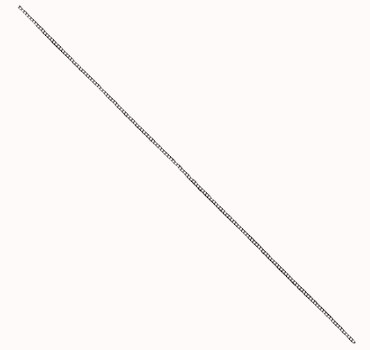
x=x+1; // Here x value increase in both conditions

getch();

}

}





# Q5. Mid-Point Circle Drawing Algorithm

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

void plot\_pts (int ,int ,int ,int); void main()

{

int gd=DETECT,gm ; int x,y,xc,yc;

ﬂoat p,r ;

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

printf("Enter the center co-ordinate :"); scanf("%d%d",&xc,&yc);

printf("Enter the radius :"); scanf("%f",&r);

x=0;

y=r; p=1.25-r ;

do{

plot\_pts(xc,yc,x,y); if(p<0)

{

p=p+((2\*x)+3);

}

else

{

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

}

x++ ;

} while(x<y);

if(x==y) plot\_pts(xc,yc,x,y); getch();

}

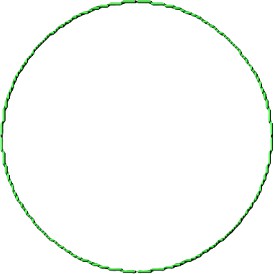
void plot\_pts(int x,int y,int x1,int y1)

{ delay(50);

putpixel(x+x1,y+y1,GREEN); putpixel(x-x1,y+y1,GREEN); putpixel(x+x1,y-y1,GREEN); putpixel(x-x1,y-y1,GREEN); putpixel(x+y1,y+x1,GREEN); putpixel(x-y1,y+x1,GREEN); putpixel(x+y1,y-x1,GREEN); putpixel(x-y1,y-x1,GREEN);

}





# Q6. 2D Translation

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

int gd = DETECT, gm;

int n, xs[100], ys[100], i, tx, ty;

void draw();

void translate();

void main() {

// Input number of sides of the polygon printf("Enter number of sides of polygon: "); scanf("%d", &n);

// Input the coordinates for each vertex printf("Enter coordinates (x, y) for each vertex:\n"); for (i = 0; i < n; i++) {

printf("Vertex %d: ", i + 1); scanf("%d%d", &xs[i], &ys[i]);

}

// Input translation distances

printf("Enter distance for translation (in x and y direction): "); scanf("%d%d", &tx, &ty);

// Initialize graphics mode

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

cleardevice();

// Draw the original polygon in RED setcolor(RED);

draw();

// Perform translation translate();

// Draw the translated polygon in GREEN setcolor(GREEN);

draw();

getch();

}

// Function to draw the polygon void draw() {

for (i = 0; i < n; i++) {

line(xs[i], ys[i], xs[(i + 1) % n], ys[(i + 1) % n]);

}

}

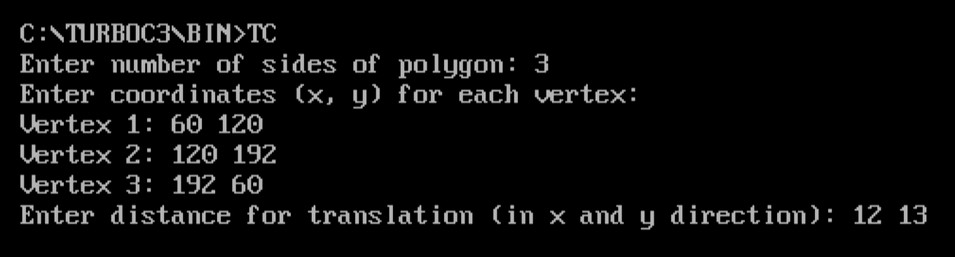
// Function to perform translation void translate() {

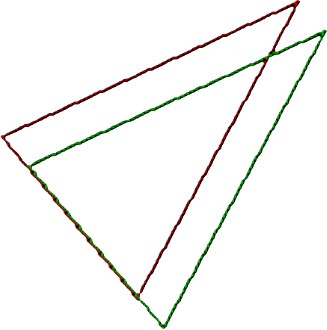
for (i = 0; i < n; i++) { xs[i] += tx;

ys[i] += ty;

}

}





# Q7. 2D Scaling

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

int gd = DETECT, gm; int n, x[100], y[100], i; ﬂoat sx, sy;

void draw(); void scale();

void main() {

// Input number of sides of the polygon printf("Enter number of sides of the polygon: ");

scanf("%d", &n);

// Input the coordinates of the vertices printf("Enter coordinates (x, y) for each vertex:\n"); for (i = 0; i < n; i++) {

printf("Vertex %d: ", i + 1); scanf("%d%d", &x[i], &y[i]);

}

// Input scaling factors

printf("Enter scaling factors (sx, sy): "); scanf("%f%f", &sx, &sy);

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

cleardevice();

// Draw the original polygon in RED setcolor(RED);

draw();

// Perform scaling scale();

// Draw the scaled polygon in GREEN setcolor(GREEN);

draw();

getch();

}

// Function to draw the polygon void draw() {

for (i = 0; i < n; i++) {

line(x[i], y[i], x[(i + 1) % n], y[(i + 1) % n]);

}

}

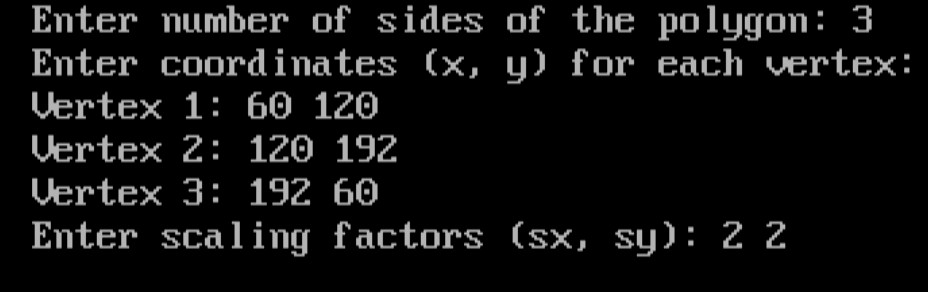
// Function to perform scaling void scale() {

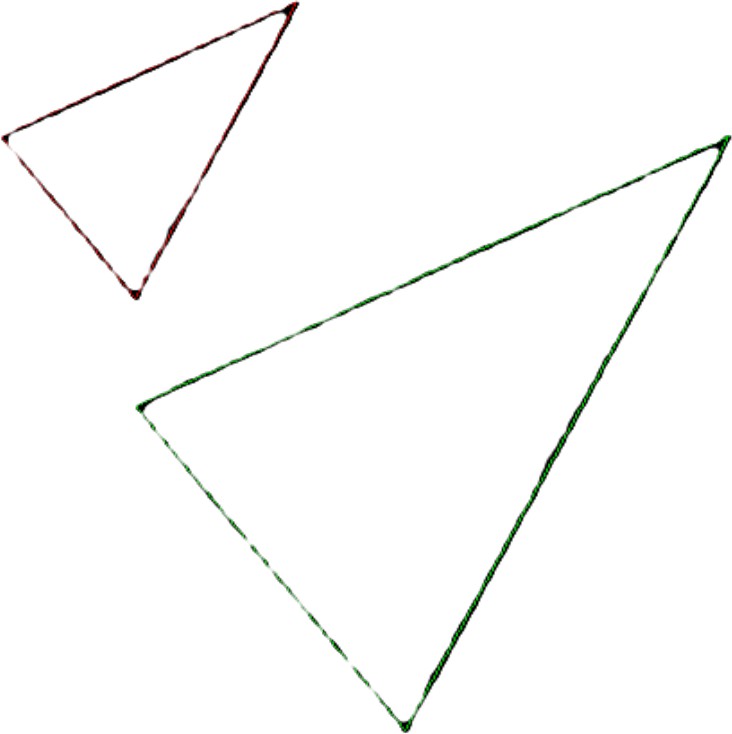
for (i = 0; i < n;i++) { x[i]=x[i]\* sx;

y[i]=y[i]\* sy;

}

}





# Q8. 2D Rotation

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

int gd = DETECT, gm;

int n, xs[100], ys[100], i, xPivot, yPivot; ﬂoat angleRad ,angleDeg;

// Function prototypes void draw();

void rotate();

void main() {

// Input number of sides of the polygon printf("Enter number of sides of the polygon: "); scanf("%d", &n);

// Input the coordinates of the vertices

printf("Enter coordinates (x, y) for each vertex:\n"); for (i = 0; i < n; i++) {

printf("Vertex %d: ", i + 1); scanf("%d%d", &xs[i], &ys[i]);

}

// Input pivot point for rotation printf("Enter pivot point (xPivot, yPivot): "); scanf("%d%d", &xPivot, &yPivot);

// Input rotation angle in degrees

printf("Enter rotation angle (in degrees): "); scanf("%f", &angleDeg);

angleRad = angleDeg \* (M\_PI / 180.0); // Convert degrees to radians

// Initialize graphics mode

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

cleardevice();

// Draw the original polygon in RED setcolor(RED);

draw();

// Perform rotation rotate();

// Draw the rotated polygon in GREEN setcolor(GREEN);

draw();

getch();

}

// Function to draw the polygon void draw() {

for (i = 0; i < n; i++) {

line(xs[i], ys[i], xs[(i + 1) % n], ys[(i + 1) % n]);

}

}

// Function to perform rotation void rotate() {

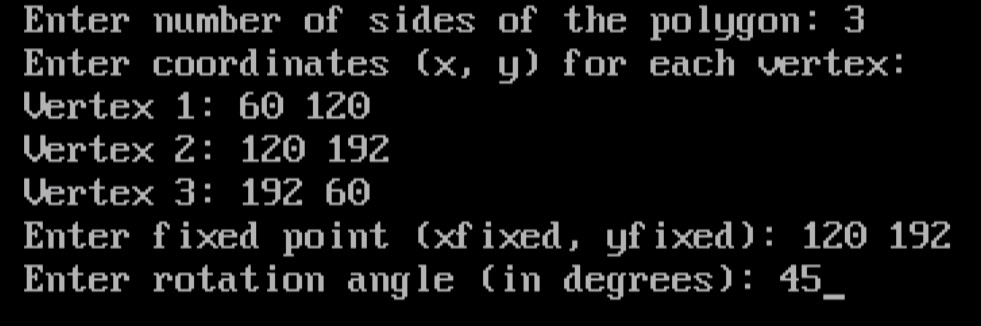
for (i = 0; i < n; i++) {

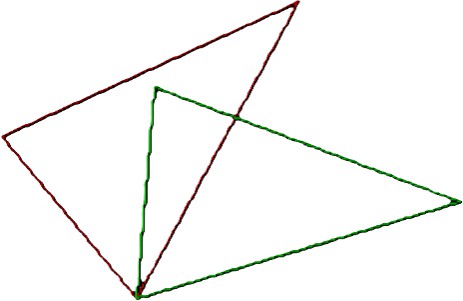
int xTemp = xs[i] - xPivot; int yTemp = ys[i] - yPivot;

xs[i] = xPivot + (xTemp \* cos(angleRad) - yTemp \* sin(angleRad)); ys[i] = yPivot + (xTemp \* sin(angleRad) + yTemp \* cos(angleRad));

}

}





# Q9. 2D Reﬂection

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

int gd = DETECT, gm;

int n, xs[100], ys[100], i; char axis;

int midX,midY;

void drawPolygon(int xs[], int ys[], int n, int color); void reﬂectPolygon(int xs[], int ys[], int n, char axis);

void main() {

printf("Enter number of sides of the polygon: "); scanf("%d", &n);

printf("Enter coordinates (x, y) for each vertex:\n");

for (i = 0; i < n; i++) { printf("Vertex %d: ", i + 1); scanf("%d%d", &xs[i], &ys[i]);

}

printf("Enter axis of reﬂection (x/y): "); scanf(" %c", &axis);

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

midX = getmaxx() / 2; midY = getmaxy() / 2;

cleardevice(); setcolor(WHITE);

line(0, midY, getmaxx(), midY); // x-axis line(midX, 0, midX, getmaxy()); // y-axis

// Draw the original polygon in RED setcolor(RED);

drawPolygon(xs, ys, n, RED);

// Reﬂect the polygon reﬂectPolygon(xs, ys, n, axis);

// Draw the reﬂected polygon in GREEN setcolor(GREEN);

drawPolygon(xs, ys, n, GREEN);

getch();

// Close graphics mode closegraph();

}

// Function to draw the polygon

void drawPolygon(int xs[], int ys[], int n,int color)

{

int midX = getmaxx() / 2; int midY = getmaxy() / 2;

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

{

int x1 = midX + xs[i]; int y1 = midY - ys[i];

int x2 = midX + xs[(i + 1) % n]; int y2 = midY - ys[(i + 1) % n]; line(x1, y1, x2, y2);

}

}

// Function to reﬂect the polygon

void reﬂectPolygon(int xs[], int ys[], int n, char axis) {

if (axis == 'x' || axis == 'X') { for (i = 0; i < n; i++) {

ys[i] = -ys[i]; // Reﬂect about x-axis

}

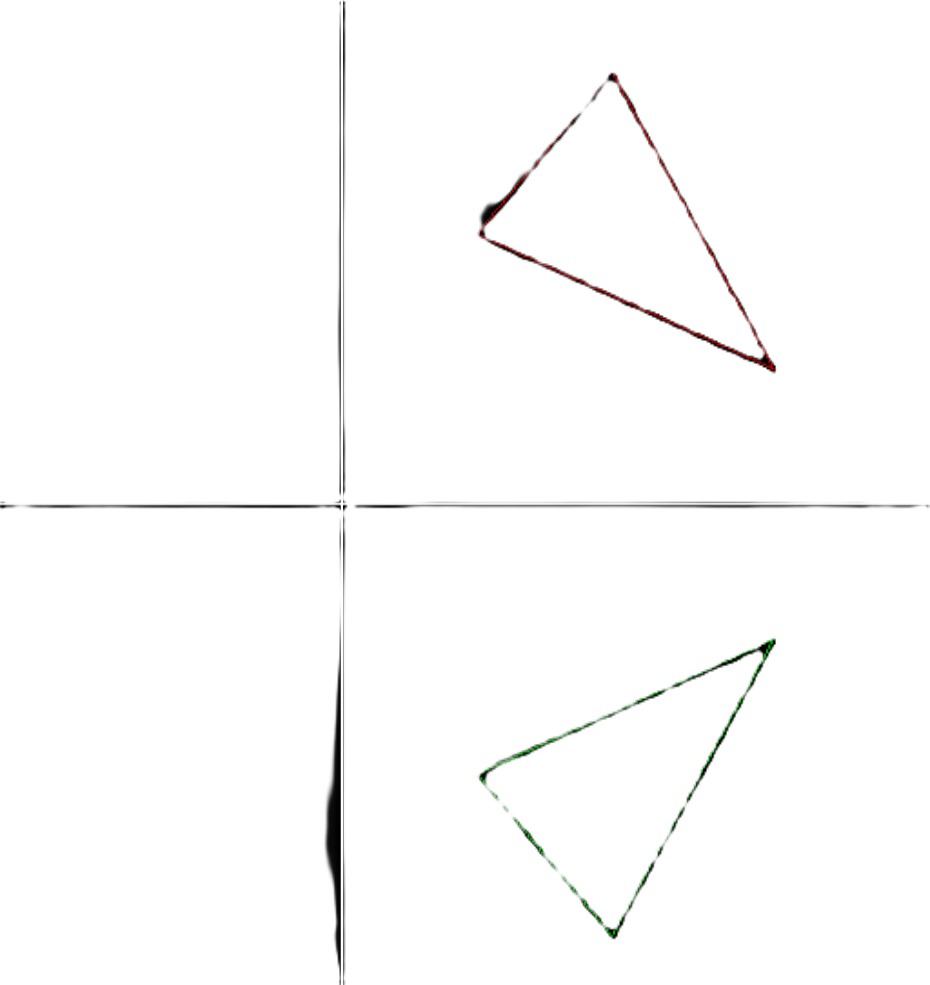
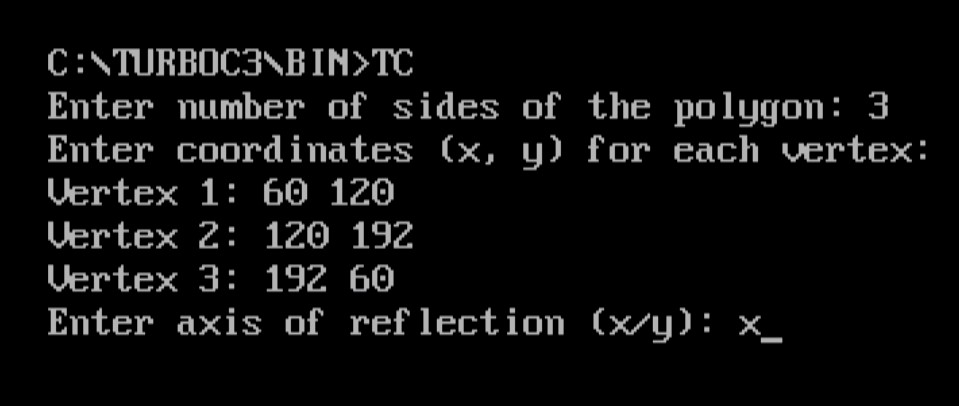
} else if (axis == 'y' || axis == 'Y') { for (i = 0; i < n; i++) {

xs[i] = -xs[i]; // Reﬂect about y-axis

}

}

}



# Q10. ﬂood\_ﬁll

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

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

void ﬂood\_Fill(int x, int y, int ﬁll\_Color, int old\_Color)

{

if (getpixel(x, y) == old\_Color)

{

putpixel(x, y, ﬁll\_Color); // Set the pixel to the ﬁll color

// To ﬁll surrounding pixels

ﬂood\_Fill(x + 1, y, ﬁll\_Color, old\_Color); // Right side ﬂood\_Fill(x - 1, y, ﬁll\_Color, old\_Color); // Left side ﬂood\_Fill(x, y + 1, ﬁll\_Color, old\_Color); // Down side ﬂood\_Fill(x, y - 1, ﬁll\_Color, old\_Color); // Up side

}

}

void main()

{

int gd = DETECT, gm;

int x, y, ﬁll\_Color, old\_Color;

initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

// A rectangle's point rectangle(100, 100, 200, 200);

// Set the starting point for ﬁlling x = 150;

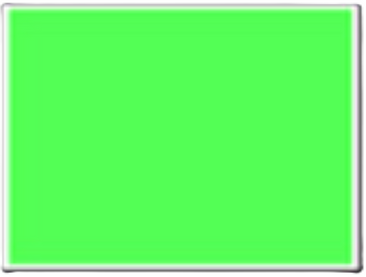
y = 150;

ﬁll\_Color = RED; old\_Color = BLACK;

ﬂood\_Fill(x, y, ﬁll\_Color, old\_Color);

getch();

}



# Q11. Boundary Fill

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

void boundaryFill(int x, int y, int ﬁllColor, int boundaryColor)

{

if (getpixel(x,y)!= boundaryColor && getpixel(x,y)!= ﬁllColor) { putpixel(x, y, ﬁllColor); // Set the pixel to the ﬁll color

delay(30);

boundaryFill(x + 1, y, ﬁllColor, boundaryColor); // Right boundaryFill(x, y-1, ﬁllColor, boundaryColor); // Left boundaryFill(x, y + 1, ﬁllColor, boundaryColor); // Down boundaryFill(x-1, y , ﬁllColor, boundaryColor); // Up

}

}

void main()

{

int gd = DETECT, gm;

int x, y, ﬁllColor, boundaryColor;

initgraph(&gd, &gm, "C:\\Turboc3\\BGI");

// Draw a closed boundary (e.g., a circle) circle(200, 200, 50);

// Set\_R the starting\_A point\_H inside the\_U boundary\_L x = 200;

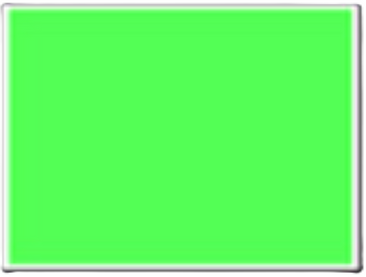
y = 200;

ﬁllColor = RED; boundaryColor = WHITE;

boundaryFill(x, y, ﬁllColor, boundaryColor);

getch();

}



# Q12. cohen-sutherland line clipping

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

int xwmax=300,xwmin=200,ywmax=100,ywmin=200,ax,ay,bx,by; void input()

{

printf("Enter TWO points (x1,y1) & (x2,y2) to Draw a line :"); scanf("%d%d%d%d",&ax,&ay,&bx,&by);

}

void draw()

{

rectangle(xwmin,ywmin,xwmax,ywmax);

}

void clip(int x,int y,int p[4])

{

if(y<ywmax) p[0]=1;

if(y>ywmin) p[1]=1;

if(x>xwmax) p[2]=1;

if(x<xwmin) p[3]=1;

else

p[3]=0;

}

void main()

{

int gd=DETECT,gm,y,x,c,p1[4],p2[4],p3[4],i; ﬂoat m; initgraph(&gd,&gm,"C:\\TURBOC3\\BGI"); cleardevice();

input(); cleardevice(); clip(ax,ay,p1);

clip(bx,by,p2); for(i=0;i<4;i++)

p3[3]=p1[i]&&p2[i]; for(i=0;i<4;i++) if(p3[i]==1)

break;

draw(); line(ax,ay,bx,by); getch(); cleardevice(); if(i!=4)

draw(); else

{

m=(ﬂoat)(by-ay)/(bx-ax); if(p1[0]==1)

y=ywmax; if(p1[1]==1)

y=ywmin; if(p1[0]==1||p1[1]==1)

{

ax=ax+(y-ay)/m; ay=y;

}

if(p2[0]==1)

y=ywmax; if(p2[1]==1)

y=ywmin; if(p2[0]==1||p2[1]==1)

{

bx=bx+(y-by)/m; by=y;

}

if(p1[2]==1)

x=xwmax; if(p1[3]==1)

x=xwmin; if(p1[2]==1||p1[3]==1)

{

ay=ay+m\*(x-ax); ax=x;

}

if(p2[2]==1)

x=xwmax; if(p2[3]==1)

x=xwmin; if(p2[2]==1||p2[3]==1)

{

by=by+m\*(x-bx); bx=x;

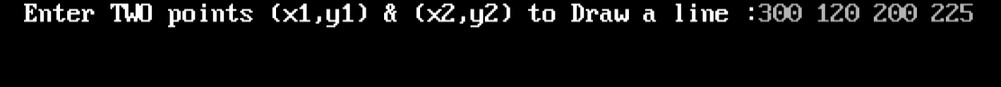
}

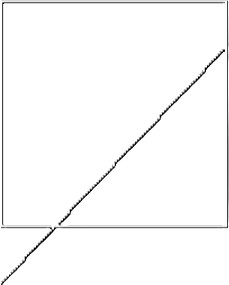
draw(); line(ax,ay,bx,by);

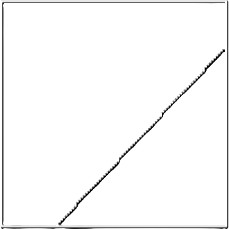
}

getch(); closegraph();

}







# Q13. Bezier Curve

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

// Line drawing function using DDA

void drawLine(int x1, int y1, int x2, int y2) { int dx, dy, steps, i;

ﬂoat xIncrement, yIncrement, x = x1, y = y1;

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

steps = (abs(dx) > abs(dy)) ? abs(dx) : abs(dy);

xIncrement = dx / (ﬂoat)steps; yIncrement = dy / (ﬂoat)steps;

for (i = 0; i <= steps; i++) { putpixel((int)x, (int)y, GREEN); x += xIncrement;

y += yIncrement; delay(50);

}

}

// bezeir curve drawing function

void drawBezierCurve(int x[], int y[]) { double putx, puty, t;

for (t = 0.0; t <= 1.0; t += 0.001) {

putx = 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];

puty = 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((int)putx, (int)puty, WHITE);

}

}

void main() {

int x[4], y[4], i;

int gd = DETECT, gm;

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

// Input points

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

printf("Enter x and y coordinates of point %d: ", i + 1); scanf("%d%d", &x[i], &y[i]);

putpixel(x[i], y[i], GREEN); // Display the points

}

// Draw lines between consecutive points for clarity for (i = 0; i < 3; i++) {

drawLine(x[i], y[i], x[i + 1], y[i + 1]);

}

// Draw the Bezier curve drawBezierCurve(x, y);

getch(); closegraph();

}

