// 1.Implement DDA Line Drawing algorithm

#include <stdio.h>

#include <stdlib.h>

#include <graphics.h>

int main()

{

int gd = DETECT, gm;

int x, y, x1, y1, x2, y2, xi, yi, k, dx, dy, s;

printf("Enter the first point(x co-ordinate): ");

scanf("%d", &x1);

printf("Enter the first point(y co-ordinate):");

scanf("%d", &y1);

printf("Enter the second point(x co-ordinate):");

scanf("%d", &x2);

printf("Enter the second point(y co-ordinate)");

scanf("%d", &y2);

initgraph(&gd, &gm, NULL);

x = x1;

y = y1;

putpixel(x, y, GREEN);

dx = x2 - x1;

dy = y2 - y1;

if (abs(dx) > abs(dy))

{

s = abs(dx);

}

else

{

s = abs(dy);

}

xi = dx / s;

yi = dy / s;

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

{

x += xi;

y += yi;

putpixel(x, y, GREEN);

}

delay(50000);

closegraph();

return 0;

}

// 2.Implement Bresenham’s Line algorithm

#include <stdio.h>

#include <graphics.h>

int main()

{

int gd = DETECT, gm;

int x, y, x1, y1, x2, y2, dx, dy, signx, signy, p, exchange;

printf("Enter the first point(x co-ordinate): ");

scanf("%d", &x1);

printf("Enter the first point(y co-ordinate):");

scanf("%d", &y1);

printf("Enter the second point(x co-ordinate):");

scanf("%d", &x2);

printf("Enter the second point(y co-ordinate)");

scanf("%d", &y2);

initgraph(&gd, &gm, NULL);

dx = abs(x2 - x1);

dy = abs(y2 - y1);

x = x1;

y = y1;

putpixel(x, y, RED);

signx = (x2 - x1) / (abs(x2 - x1));

signy = (y2 - y1) / (abs(y2 - y1));

if (dy > dx)

{

exchange = 1;

}

else

{

exchange = 0;

}

p = (2 \* dy) - dx;

while (x <= x2)

{

if (p < 0)

{

if (exchange == 1)

{

y += signy;

}

else

{

x += signx;

}

p += (2 \* dy);

}

else

{

x += signx;

y += signy;

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

}

putpixel(x, y, RED);

}

delay(5000);

closegraph();

return 0;

}

// 3.Implement midpoint Circle algorithm.

#include<stdio.h>

#include<graphics.h>

#include<stdlib.h>

int main()

{

int gd= DETECT, gm;

int x,y,r,xc,yc,p;

printf("Enter the x co-ordinate:");

scanf("%d",&xc);

printf("Enter the y co-ordinate:");

scanf("%d",&yc);

printf("Enter the radius of the circle:");

scanf("%d",&r);

if(r<=xc && r<=yc)

{

p=1-r;

x=0;

y=r;

initgraph(&gd,&gm,NULL);

while(x<y){

putpixel(xc+x, yc+y,WHITE);

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

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

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

putpixel(xc+y, yc+x,WHITE);

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

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

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

if(p<0)

p=p+2\*x+3;

else{

p=p+2\*(x-y)+5;

y=y-1;

}

x++;

delay(150);

}

delay(10000);

closegraph();

}

else{

printf("The co-ordinates are invalid");

}

return 0;

}

// 4.Implement Area Filling Algorithm using Flood Fill (4-connected)

#include <graphics.h>

#include <conio.h>

#include <stdio.h>

// Function to perform flood fill

void floodFill(int x, int y, int oldColor, int newColor) {

// Check if the current pixel is within the boundaries and has the old color

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

// Set the new color for the current pixel

putpixel(x, y, newColor);

// Recursively call floodFill for adjacent pixels

floodFill(x + 1, y, oldColor, newColor); // Right

floodFill(x - 1, y, oldColor, newColor); // Left

floodFill(x, y + 1, oldColor, newColor); // Down

floodFill(x, y - 1, oldColor, newColor); // Up

}

}

int main() {

int gd = DETECT, gm;

initgraph(&gd, &gm, "");

// Draw a rectangle with a boundary

rectangle(50, 50, 200, 200);

// Set the old color (white) and new color (red)

int oldColor = WHITE;

int newColor = RED;

// Perform flood fill starting from the top-left corner

floodFill(51, 51, oldColor, newColor);

getch();

closegraph();

return 0;

}

// 5.Implement Area Filling Algorithm using Boundary Fill

#include <stdio.h>

#include <graphics.h>

void boundary\_fill(int x, int y, int fcolor, int bcolor) {

if (getpixel(x, y) != bcolor && getpixel(x, y) != fcolor) {

putpixel(x, y, fcolor);

delay(10);

if (x + 1 < getmaxx()) boundary\_fill(x + 1, y, fcolor, bcolor);

if (y + 1 < getmaxy()) boundary\_fill(x, y + 1, fcolor, bcolor);

if (y - 1 >= 0) boundary\_fill(x, y - 1, fcolor, bcolor);

if (x - 1 >= 0) boundary\_fill(x - 1, y, fcolor, bcolor);

}

}

int main() {

int x, y, fcolor, bcolor;

int gd = DETECT, gm;

initgraph(&gd, &gm, "null");

rectangle(100, 100, 250, 300);

boundary\_fill(115, 110, 12, 15);

delay(50000);

closegraph();

return 0;

}

// 6.Implement 2D Transformations: Translation

#include <stdio.h>

#include <graphics.h>

int main() {

int gd = DETECT, gm;

int x1, y1, x2, y2, tx, ty, x3, y3, x4, y4;

printf("Enter the starting point of line segment (x1 y1): ");

scanf("%d %d", &x1, &y1);

printf("Enter the ending point of line segment (x2 y2): ");

scanf("%d %d", &x2, &y2);

printf("Enter translation distance tx and ty:\n");

scanf("%d %d", &tx, &ty);

initgraph(&gd, &gm, "NULL");

setcolor(5);

line(x1, y1, x2, y2);

x3 = x1 + tx;

y3 = y1 + ty;

x4 = x2 + tx;

y4 = y2 + ty;

setcolor(7);

line(x3, y3, x4, y4);

delay(3000);

closegraph();

return 0;

}

// 7.Implement 2D Transformations: Scaling

#include <stdio.h>

#include <math.h>

#include <graphics.h>

int main() {

int gd = DETECT, gm;

int x1, y1, x2, y2, sx, sy, x3, y3, x4, y4;

printf("Enter the starting point of line segment (x1, y1): ");

scanf("%d%d", &x1, &y1);

printf("Enter the ending point of the line segment (x2, y2): ");

scanf("%d%d", &x2, &y2);

printf("Enter the scaling distance (sx, xy): ");

scanf("%d%d", &sx, &sy);

initgraph(&gd, &gm, NULL);

setcolor(5);

line(x1, y1, x2, y2);

x3 = x1 \* sx;

y3 = y1 \* sy;

x4 = x2 \* sx;

y4 = y2 \* sy;

setcolor(7);

line(x3, y3, x4, y4);

delay(15000);

closegraph();

return 0;

}

// 8.Implement 2D Transformations: Rotation

#include <stdio.h>

#include <math.h>

#include <graphics.h>

int main() {

int gd = DETECT, gm;

int x1, y1, x2, y2, x3, y3, x4, y4;

float a, t;

printf("Enter the starting point of line segment (x1, y1): ");

scanf("%d%d", &x1, &y1);

printf("Enter the ending point of the line segment (x2, y2): ");

scanf("%d%d", &x2, &y2);

printf("Enter the angle of rotation: ");

scanf("%f", &a);

initgraph(&gd, &gm, NULL);

setcolor(5);

line(x1, y1, x2, y2);

t = a \* (3.14 / 180);

x3 = (x1\*cos(t)) - (y1\*sin(t));

y3 = (x1\*sin(t)) + (y1\*cos(t));

x4 = (x2\*cos(t)) - (y2\*sin(t));

y4 = (x2\*sin(t)) + (y2\*cos(t));

setcolor(7);

line(x3, y3, x4, y4);

delay(15000);

closegraph();

return 0;

}

// 9.Implement 3D Transformations: Translation

#include <stdio.h>

#include <math.h>

#include <graphics.h>

int main() {

int gd = DETECT , gm;

int a1, b1, a2, b2, dep, x, y;

int x1, y1, x2, y2, depth;

printf("3D Translation:-\n\n");

printf("Enter 1st to value (x1, y1): ");

scanf("%d%d", &x1, &y1);

printf("Enter the bottom value (x2, y2): ");

scanf("%d%d", &x2, &y2);

printf("Enter the Translation Distances (x, y): ");

scanf("%d%d", &x, &y);

initgraph(&gd, &gm, NULL);

depth = (x2 - x1) / 4;

bar3d(x1, y1, x2, y2, depth, 1);

a1 = x1 + x;

a2 = x2 + x;

b1 = y1 + y;

b2 = y2 + y;

dep = (a2-a1)/4;

bar3d(a1, b1, a2, b2, dep, 1);

delay(20000);

closegraph();

return 0;

}

// 10.Implement 3D Transformations: Scaling

#include <stdio.h>

#include <math.h>

#include <graphics.h>

int main() {

int gd = DETECT , gm;

int a1, b1, a2, b2, dep, x, y;

int x1, y1, x2, y2, depth;

printf("3D Scaling:-\n\n");

printf("Enter 1st to value (x1, y1): ");

scanf("%d%d", &x1, &y1);

printf("Enter the bottom value (x2, y2): ");

scanf("%d%d", &x2, &y2);

printf("Enter the Scaling Distances (x, y): ");

scanf("%d%d", &x, &y);

initgraph(&gd, &gm, NULL);

depth = (x2 - x1) / 4;

bar3d(x1, y1, x2, y2, depth, 1);

a1 = x1 \* x;

a2 = x2 \* x;

b1 = y1 \* y;

b2 = y2 \* y;

dep = (a2-a1)/4;

bar3d(a1, b1, a2, b2, dep, 1);

delay(20000);

closegraph();

return 0;

}

// 11.Implement Curve: Bezier curve

#include <stdio.h>

#include <math.h>

#include <graphics.h>

int main()

int x[4], y[4];

int i;

double t;

int gd = DETECT, gm;

printf("Enter the X and Y Co ordinate of the four control points: ");

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

{

scanf("%d %d", &x[i], &y[i]);

}

initgraph(&gd, &gm, NULL);

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

{

putpixel(x[i], y[i], YELLOW);

}

for (t = 0.0; t < 1.0; t += 0.0005)

{

double xt = pow(1 - t, 3) \* x[0] + 3 \* t \* pow(1 - t, 2) \* x[1] + 3 \* pow(t, 2) \* (1 - t) \* x[2] + pow(t, 3) \* x[3];

double yt = pow(1 - t, 3) \* y[0] + 3 \* t \* pow(1 - t, 2) \* y[1] + 3 \* pow(t, 2) \* (1 - t) \* y[2] + pow(t, 3) \* y[3];

putpixel((int)xt, (int)yt, WHITE);

}

delay(15000);

closegraph();

return 0;

}